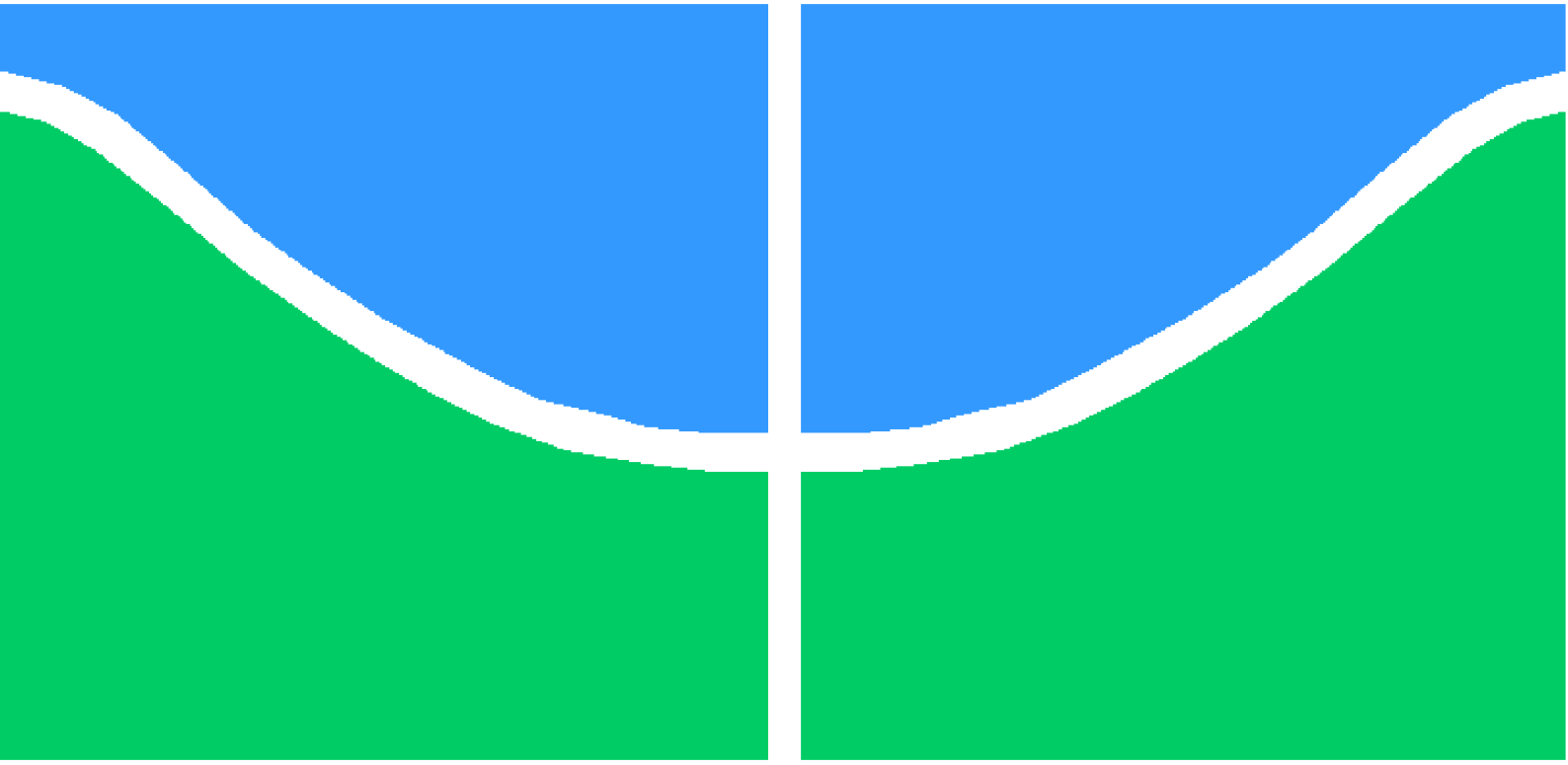


Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Journpic: desenvolvimento de um diário fotográfico para dispositivos iOS

Autor: Danilo Maia Rodrigues
Orientador: Prof. Dr. André Barros de Sales

Brasília, DF
2015

Danilo Maia Rodrigues

Journpic: desenvolvimento de um diário fotográfico para dispositivos iOS

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. André Barros de Sales

Brasília, DF

2015

Danilo Maia Rodrigues
Journpic: desenvolvimento de um diário fotográfico para dispositivos iOS/ Danilo Maia Rodrigues. – Brasília, DF, 2015-
94 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. André Barros de Sales

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2015.

1. Journpic. 2. iOS. I. Prof. Dr. André Barros de Sales. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Journpic: desenvolvimento de um diário fotográfico para dispositivos iOS

CDU 02:141:005.6

Danilo Maia Rodrigues

Journpic: desenvolvimento de um diário fotográfico para dispositivos iOS

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Prof. Dr. André Barros de Sales
Orientador

Profa. MSc. Fabiana Freitas Mendes
Membro

Profa. Dra. Carla Silva Rocha Aguiar
Membro

Prof. Phelippe Amorim
Membro

Brasília, DF
2015

Agradecimentos

Agradeço ao Dennis Merli, ao Paulo Magalhães, ao Phelippe Amorim e aos demais integrantes do projeto BEPiD, por colaborarem direta ou indiretamente com módulos do desenvolvimento deste trabalho.

Resumo

Este trabalho apresenta o desenvolvimento do Journpic, um aplicativo para plataforma iOS implementado na linguagem Objective-C. O Journpic é um diário fotográfico que oferece filtros de busca por tag e emoção atrelada à foto. Ele, portanto, foca nos padrões de usabilidade de interface, assim como a solução de problemas de âmbito social e cognitivo observados durante a utilização de dispositivos móveis. Para isto, o aplicativo teve a acessibilidade e a interface gráfica como pontos de maior importância durante o desenvolvimento. O Journpic foi construído aproveitando várias das ferramentas proporcionadas pela IDE de desenvolvimento XCode, como análise da carga de processamento, consumo de memória, vazamentos de memória e alocações de classes em tempo real, com o propósito de atingir um software de qualidade. O aplicativo foi publicado na *App Store*, loja de aplicativos da Apple.

Palavras-chaves: iOS. Usabilidade. Acessibilidade. Diário fotográfico.

Abstract

This work presents the development of Journpic, an app for iOS platform written in Objective-C programming language. Journpic is a photographic journal that offers filters by tag and emotion related to the photo. Thus, it focus user interface guidelines, as well as solutions for social and cognitive problems observed during mobile devices utilization. To achieve this, this app had accessibility and graphic interface as leading points during the development stage. Journpic was developed using many of the tools made avaiable by XCode IDE, such as processing workload, memory usage, memory leaks a real time class allocations, with the purpose to achieve a quality software. The app was published in App Store.

Key-words: iOS. Usability. Accessability. Photographic journal.

Lista de ilustrações

Lista de tabelas

Lista de algoritmos

Lista de abreviaturas e siglas

API	*Application Programming Interface*
BEPiD	Programa Educacional Brasileiro de Desenvolvimento para iOS
GB	*GigaBytes*
GHz	*GigaHertz*
IHC	Interação Humano-Computador
MVC	*Model View Controller*
OO	Orientado a Objeto
RUP	*Rational Unified Process*
UCB	Universidade Católica de Brasília
UML	*Unified Modelling Language*
UnB	Universidade de Brasília
XIB	*NeXT Interface Builder*

Sumário

1 Introdução

1.1 Contexto

A evolução do mercado de Tecnologia da Informação (TI) contribuiu continuamente para a melhoria de processos e serviços, agregando valor a eles (TORRES, 1995). Os computadores pessoais se popularizaram com o tempo e, com eles, cresce o número e a variedade de dispositivos.

Montoliu e Gatica-Perez (2010) conseguiram observar que a popularização e evolução de dispositivos móveis tem feito com que estes se revelem substitutos para os aparelhos de multimídia em geral naturalmente, dado que são capazes de acessar, manejar e enviar tipos de mídia distintos como localização, fotos, vídeos e áudios. Estes autores também identificaram a necessidade de pesquisas referente a interfaces com o usuário que provenha usabilidade e de estudos referentes à análise do comportamento dos usuários.

A aplicação de usabilidade em dispositivos móveis é singular em relação às demais, pois possuem limitações que são ausentes em computadores pessoais. A limitação óbvia é relativo ao aspecto físico. Por exemplo, um computador pessoal tem a tela várias vezes maior quando comparado à tela de um dispositivo móvel. Neste caso, vale ressaltar que dois fatores são considerados quando se tem a tela de um dispositivo móvel como fator importante para a usabilidade de interface: o tamanho diagonal da tela do dispositivo, que define quantas polegadas o mesmo possui; e a quantidade de pixels contidos nesta tela, ou seja, a resolução da tela.

Oinas-kukkonen (2003) constatou durante sua pesquisa acerca do desenvolvimento de aplicativos para dispositivos móveis que, os bem sucedidos não apenas possuem um design que se adéqua à limitação física do tamanho da tela, mas também em outras restrições desta categoria de aparelhos, tais como as reduzidas bandas de acesso, esporádicas interrupções no processamento e na memória, e diferença entre a maneira de inserção de dados em geral, quando comparados aos computadores convencionais. Além das limitações citadas, há outros aspectos igualmente importantes que devem ser observados, mas que pertencem a outros contextos, como o cognitivo e o social.

Considerando uma abordagem cognitiva, é comprovado cientificamente que o cérebro humano é muito suscetível a interferências externas, que fazem com que um usuário perca sua atenção enquanto utiliza um dispositivo móvel. Em outras palavras, ao interagir com um aplicativo, o usuário está propenso a perder sua atenção devido a eventos ocor-

rendo no meio e essa desconcentração afeta na velocidade de interação com o aplicativo e também influencia o tempo necessário para a aprendizagem (GAZZALEY, 2012). Este tipo de fenômeno pode ser observado com um passageiro de metrô, o qual precisa prestar atenção no nome da estação dita pela cabine de controle enquanto utiliza um dispositivo móvel.

Do ponto de vista social, pode-se encontrar as limitações principalmente na área referente às diferentes culturas das pessoas de diferentes localidades, mas que podem vir a utilizar as mesmas interfaces. Usuários de diferentes lugares normalmente têm ideias e expectativas distintas referentes aos aplicativos que irão utilizar (GALITZ, 2007). Por exemplo, pessoas na América do Sul podem interpretar um ícone de hipopótamo para um aplicativo como algo fofo, já as pessoas de comunidades ribeirinhas na África Central podem interpretar diferente, já que o índice de mortes nestes locais devido a ataques deste animal é muito alto. A própria finalidade da aplicação pode gerar expectativas distintas de usuário para usuário.

De maneira geral, pode-se afirmar que, com o aumento no número e na variedade dos dispositivos móveis, é gerado um grande desafio àqueles que trabalham no campo de Interação Humano-Computador (IHC), dado que os usuários esperam que os diversos aplicativos que utilizam possam ser acessados facilmente em quaisquer dispositivo móvel que ele tenha, ainda que os sistemas operacionais e configurações de hardware, como tamanho de tela, capacidade de memória, armazenamento e processamento mudem bastante de dispositivo para dispositivo. É em face a este desejo do usuário que este projeto visa desenvolver um aplicativo que contemple os casos citados.

O termo "usabilidade[1]" tem se tornado mais comum e ganhou destaque ao longo dos anos. Isto implicou a motivação para desenvolvimento de interfaces mais aprimoradas para sistemas e computadores em geral.

Há anos atrás os textos científicos possuíam poucos estudos que conectavam o progresso na área de *desing* de interfaces com o campo de usabilidade em dispositivos móveis. Esta falta de estudos levou os desenvolvedores a seguirem heurísticas gerais (como a visibilidade do estado do sistema, complacência entre sistema e mundo real, controle e liberdade do usuário, etc.) que foram propostas por Nielsen (1994). A falta de conexão específica entre as duas áreas fez com que as aplicações para dispositivos móveis em geral ainda possuíssem problemas de usabilidade naquela época.

Anos depois, o crescente mercado de aplicativos móveis capazes de executar aplicações desenvolvidas por terceiros, os smartphones, gerou uma concorrência constante para o mercado de aplicativos, os quais ainda hoje têm que melhorar continuamente para sobreviver. O aumento da experiência do usuário é a chave para a melhora dos aplicativos

[1] Usabilidade - Usabilidade é um atributo de qualidade que avalia a facilidade de utilização de interfaces de usuário (NIELSEN, 1994)

(NUNES; CORREIA, 2013).

A disseminação e popularização dos dispositivos móveis contribuiu para a diversificação dos modelos de cada um desses dispositivos, os quais apresentam vários aspectos diferentes, como o sistema operacional, disposição dos botões no aparelho, tamanho de tela, processamento, etc. Face a isto, as empresas que desenvolveram estes aparelhos fornecem guias e indicações de uso para o desenvolvimento das interfaces gráficas em seus aparelhos, com o propósito de garantir a usabilidade nos aplicativos desenvolvidos para seus respectivos dispositivos móveis. Algumas dessas empresas estão:

- Apple, responsável pelo sistema iOS.
- Blackberry, responsável pelo sistema Blackberry.
- Google, responsável pelo sistema Android.
- Microsoft, responsável pelo sistema Windows Phone.

Os aplicativos criados pelas empresas que desenvolvem os sistemas operacionais dos dispositivos móveis também estão sujeitos à críticas e apresentam melhorias a cada nova versão. Estes dispositivos móveis normalmente possuem câmeras, e portanto os sistemas operacionais apresentam um aplicativo que permite o usuário tirar fotos ou visualizá-las. Contudo, os aplicativos de galerias de fotos não possuem muitos filtros de busca, o que em alguns casos só permite o usuário filtrar fotos somente por data e álbum.

A Tabela 1 demonstra as opções de filtro de busca presentes em cada um dos respectivos aplicativos de visualização do fotos dos sistemas operacionais citados anteriormente.

Sistema operacional	**Filtros de busca de fotos**
iOS	Data, local, álbum e favoritos
Blackberry	Data e álbum
Android	Data, local e álbum
Windows Phone	Data, álbum e favoritos

Tabela 1: Filtros de busca dos visualizadores de fotos de cada um dos sistemas operacionais

É constatado a partir da Tabela 1 que não há filtros de busca que contemplem outros aspectos, como a emoção sentida no momento da foto ou mesmo rótulos para as imagens, nos aplicativos de visualização de fotos padrão de cada um dos sistemas.

A disponibilização dos guias de interface mostra que a usabilidade ultrapassou o campo de pesquisas científicas e agora está atrelado ao sucesso de mercado, mas ainda que estes guias possuam finalidades semelhantes, têm termos distintos para seus elementos

de interface, e estes elementos podem variar de posição dependendo do sistema no qual estão contidos. Face a diversidade, este trabalho de conclusão de curso estará focado nos dispositivos móveis com sistema iOS, visando a implementação de funcionalidades de busca que não estejam presentes nos demais aplicativos de visualização de fotos (filtro por rótulo e por emoção).

1.2 Problematização

Os aplicativos de visualização de fotos que vêm integrados com os sistemas operacionais para dispositivos móveis possuem filtros de busca bastante limitados e que não contemplam os aspectos sociais e cognitivos descritos anteriormente.

1.3 Objetivos

1.3.1 Objetivo geral

Desenvolver um aplicativo de fotos para um dispositivo móvel com novas funcionalidades de busca.

1.3.2 Objetivos específicos

- Investigar as principais heurísticas de usabilidade.
- Investigar os padrões de usabilidade utilizados no sistema operacional.
- Investigar o aplicativo de fotos nativo do sistema operacional.
- Compreensão de como aplicar conceitos de Engenharia de Software no desenvolvimento do aplicativo.
- Implementar o aplicativo.

1.4 Metodologia

A metodologia deste trabalho foi dividida em algumas partes sendo elas: fase de investigação, fase de modelagem e fase de desenvolvimento. Cada uma destas fases abrange atividades que foram realizadas ao longo do projeto com o objetivo de atingir o sucesso com o aplicativo finalizado.

A primeira fase foi a de investigação. Nesta fase foram realizadas pesquisas sobre as principais heurísticas de usabilidade existentes e como elas se aplicam a dispositivos móveis. Posteriormente, foram investigadas as regras de interface do sistema operacional,

visando a concordância entre essas e o aplicativo a ser desenvolvido. Por fim, foi investigado o aplicativo visualizador de fotos padrão do sistema operacional, visando identificar quais são os filtros de busca de fotos que ele possui.

A segunda foi a fase de modelagem. Nesta fase foi realizada a concepção inicial do projeto, checando a viabilidade do mesmo. As atividades que compuseram esta fase estavam relacionadas ao levantamento de informações sobre trabalhos correlatos, requisitos iniciais, restrições e regras de negócio, e definição tanto da solução tecnológica adotada no desenvolvimento quanto da metodologia de projeto. Para a realização destas atividades foram utilizados conceitos aprendidos durante o curso de Engenharia de Software.

A fase final foi a de construção. Esta fase foi dedicada à implementação do aplicativo proposto, seguindo a metodologia e considerando as informações levantadas durante a fase de modelagem. A fase de construção foi composta de atividades que abrangiam o desenvolvimento desde as etapas iniciais, como a escolha de um repositório remoto, à verificação e validação das funcionalidades do aplicativo em sua versão final, descritos no Capítulo 4. Durante esta fase foram aplicados os conhecimentos de usabilidade investigados previamente.

A escrita do documento do trabalho de conclusão de curso foi feito ao longo das etapas citadas.

1.5 Organização do trabalho

Este trabalho está separado em seis capítulos, incluindo o presente.

O Capítulo 2 apresenta conceitos de usabilidade e suas principais heurísticas. Neste capítulo é possível entender a importância deste tópico e quais de seus conceitos foram aplicados ao sistema finalizado.

No Capítulo 3, é apresentada a proposta do sistema, abordando os requisitos e restrições do mesmo, assim como recursos necessários para que o projeto ocorra. Este capítulo visa apresentar ao leitor uma visão geral dos principais aspectos e funcionalidades trabalhados posteriormente no desenvolvimento.

No Capítulo 4, é apresentado ao leitor o planejamento do projeto. Este capítulo aborda aspectos mais detalhados, como o ciclo de vida do projeto e como foram determinadas as tecnologias de implementação. Planos gerenciais, necessários para um desenvolvimento adequado, também foram descritos também neste capítulo.

Posteriormente, neste mesmo capítulo, são explicitados detalhes da implementação do projeto, oferecendo ao leitor informações sobre como foram tratados o controle de versão, a implementação do modelo arquitetural, a internacionalização, a adaptabilidade da interface gráfica nos demais dispositivos, entre outros. Trechos de código de

algumas funcionalidades foram inseridos para explicar, minuciosamente, como foi dada a implementação.

Os dois últimos capítulos discutem os pontos de maior importância deste projeto, apresentando os resultados e conclusões deste trabalho.

2 Aspectos Gerais de Usabilidade

Neste capítulo são mostrados as principais componentes de qualidade e heurísticas de usabilidade definidas por Nielsen (1994), considerado o pai da usabilidade, e como elas foram tratadas pelo aplicativo fruto deste trabalho de conclusão de curso.

2.1 Definições de usabilidade

Segundo Nielsen (2012), usabilidade é um atributo de qualidade que determina o quão facilmente uma interface de usuário pode ser utilizada. A palavra "usabilidade" também se refere a métodos para aprimorar a facilidade de uso durante o processo de design. Este autor diz que usabilidade pode ser definida por cinco componentes de qualidade:

1. Apreensibilidade - O quão fácil é para um usuário concluir uma tarefa durante a primeira vez que se deparam com o design.
2. Eficiência - Uma vez que os usuários aprenderam o design, o quão rápido eles conseguem realizar as tarefas.
3. Memorização - Quando usuários retornam ao design depois de um período sem utilizá-lo, quão facilmente eles podem restabelecer a proficiência.
4. Erros - Quantos erros os usuários cometem, quão severo são esses erros, e quão facilmente eles podem se recuperar desses erros.
5. Satisfação - O quão agradável é de se utilizar o design.

Nielsen (2012) também diz que existem inúmeros vários outros atributos de qualidade importantes. Um atributo chave é utilidade, que se refere a funcionalidade de design: o sistema faz o que o usuário precisa?

- Utilidade - Se provê as funcionalidades esperadas.
- Usabilidade - Quão fácil e agradável essas funcionalidades são de usar.
- Útil - Contempla a usabilidade e a utilidade simultaneamente.

Usabilidade e utilidade são igualmente importantes e juntos determinam se algo é útil: Pouco importar se algo é fácil de ser utilizado se não realiza o esperado. Também

não é ideal se um sistema pode hipoteticamente realizar o que o usuário precisa, mas este mesmo usuário não consegue realizar a ação porque a interface é muito difícil. Para aferir a utilidade de um design, podem-se utilizar os mesmos métodos de pesquisa para usuários que ajudam a aprimorar a usabilidade (NIELSEN, 2012).

2.2 Heurísticas de usabilidade

Existem orientações para desenvolver um design adequado. Nielsen (1994) lista dez princípios gerais para design de interação.

1. Visibilidade do estado do sistema - O sistema deve sempre manter os usuários informados sobre o que está ocorrendo, através de *feedback* apropriado em tempo sensato.

2. Compatibilidade entre o sistema e o mundo real - O sistema deve falar a língua do usuário, com palavras, frases e conceitos similares ao usuário, ao invés de utilizar terminologias do sistema. Siga convenções do mundo real, fazendo com que as informações apareçam em ordem natural e lógica.

3. Controle e liberdade do usuário - Usuários geralmente escolhem funcionalidades do sistema por engano e necessitam de uma ”saída de emergência“ claramente apontada para deixar o estado indesejado sem ter que passar por um processo extenso. Dê suporte a ações de desfazer e refazer.

4. Consistência e padrões - Usuários não devem ter que adivinhar se diferentes palavras, situações, ou ações significam a mesma coisa. Siga as convenções da plataforma.

5. Prevenção de erros - Ainda melhor que boas mensagens de erro é um desing cuidado que previne que erros aconteçam em primeiro lugar. Ou elimine condições suscetíveis a erro ou cheque por eles e apresente ao usuário uma opção de confirmação antes de executarem a ação.

6. Reconhecimento ao invés de recordação - Minimize os a carga de informações que o usuário precisa guardar fazendo com que objetos, ações e opções estejam visíveis. Um usuário não deve ter que se lembrar da informação de uma parte do processo para a outra. Instruções para o uso adequado do sistema devem estar sempre visíveis ou facilmente alcançáveis sempre que apropriado.

7. Flexibilidade e eficiência de uso - Atalhos (que passam despercebidos pelo usuário novato) geralmente pode acelerar o processo de interação para o usuário avançado de maneira que o sistema possa atender tanto usuários inexperientes e experientes. Permita que usuários customizem ações frequentes.

8. Estética e design minimalista - Os processos que o usuário executa não devem conter informações irrelevantes ou que são raramente necessárias. Cada unidade extra de informação visível ao usuário compete com as unidades relevantes de informação e diminui sua visibilidade relativa.

9. Ajude usuários a reconhecer, diagnosticar e se recuperar de erros - Mensagens de erro devem ser expressas em linguagem simples (sem códigos), precisamente indicar o problema, e construtivamente sugerir uma solução.

10. Ajuda e documentação - Ainda que seja melhor que o sistema possa ser utilizado sem documentação, pode ser necessário prover ajuda e documentação. Qualquer tipo de informação deve ser fácil de ser pesquisada, focada nas tarefas do usuário, listar passos concretos a serem seguidos, e não serem muito grandes.

Estes princípios são nomeados de heurísticas por serem regras amplas, boas práticas, e não especificamente *guidelines* de usabilidade. A utilização destas regras amplas se deve a necessidade de manter a usabilidade nas aplicações, como comentado na contextualização deste trabalho (Seção 1.1).

2.3 Aprimoramento da usabilidade

Em seu artigo sobre introdução a usabilidade, no tópico que aborda como aprimorá-la, Nielsen (2012) cita a existência de vários métodos para estudar usabilidade (que não foram explicitados), mas a mais fácil e útil são os testes de usuário, os quais possuem três componentes. A Figura 1 traz a relação entre estes componentes.

Figura 1: Ciclo de prototipação

A Figura 1 mostra a relação cíclica entre os principais passos da prototipação de interfaces, descritos a seguir:

- Desenhar protótipos de interface.
- Conseguir usuários representativos, como clientes para um site de comércio eletrônico ou empregados para uma intranet, e pedir que estes usuários cumpram tarefas representativas com o dado design.
- Observar o que os usuários fazem, onde eles têm êxito, e onde eles têm dificuldade com a interface.

Nielsen (2012) enfatiza que é importante testar os usuários individualmente e deixar que resolvam quaisquer problemas por si só. Se o condutor dos testes ajudá-los o direcionar a atenção para qualquer parte em particular da tela, os resultados do teste foram contaminados.

2.4 Considerações

Este capítulo visou apresentar os principais conceitos e heurísticas de usabilidade. Para o desenvolvimento deste projeto, foram utilizadas todas as heurísticas de usabilidade apresentadas. O detalhamento de como cada uma delas foi utilizada pode ser visto nos resultados de trabalho de conclusão de curso (Seção 5.2).

3 Proposta do sistema

Seguindo os conceitos de Engenharia de Software, este capítulo visa apresentar os requisitos do aplicativo e os recursos de software e hardware necessários para o desenvolvimento do aplicativo.

3.1 Requisitos do sistema proposto

A partir dos objetivos apresentados, esta proposta consiste na criação de um aplicativo de diário fotográfico que possua filtros de busca ausentes nativamente no sistema operacional iOS e que foque as heurísticas de usabilidade apresentadas no Capítulo 2. A implementação destes filtros permite que o usuário encontre mais rapidamente suas fotos ao diminuir o volume de itens apresentados.

Considerando o temas de design de interfaces, é importante lembrar que há vários dispositivos de tamanhos distintos com o sistema iOS e que os usuários possuem características físicas e cognitivas diferentes, ainda que exerçam as mesmas tarefas no aplicativo.

Face ao exposto, os principais tópicos abordados durante o desenvolvimento deste trabalho de conclusão de curso, assim como uma breve descrição dos mesmos foram enumerados a seguir:

1. **Busca de fotos** - Qualquer usuário pode ter o desejo de ter um diário. Ainda que várias das pessoas não possuam uma boa memória para lembrar de datas, a emoção ou o lugar normalmente são bem fáceis de serem relembrados. Portanto, este aplicativo deve contar com a opção de atrelar palavras-chave, cores e emoções a cada uma das entradas do diário para facilitar uma posterior busca através deste filtros.

2. **Adaptabilidade** - Existem diferentes dispositivos móveis que têm o sistema operacional iOS, e assim existem diferentes tamanhos de tela. O aplicativo desenvolvido deve usar restrições para a interface gráfica, os quais garantirão que todos os elementos de interface tenham o tamanho e a apresentação adequada em quaisquer telas que sejam apresentados.

3. **Área de toque** - É ingenuidade imaginar que todos os usuários interagem igualmente com o aplicativo. Todos eles possuem características distintas, mas há uma

em específico que vale a pena ser mencionada: o tamanho do dedo. Usuários com o dedo muito grosso tocam em uma área muito grande da tela e com isso podem acabar selecionando opções que não gostariam por serem muito pequenas e difíceis de selecionar, e isto fatalmente irá frustrar o usuário. O aplicativo deve adotar um tamanho mínimo de 44 pontos para todo e qualquer elemento gráfico que possa ter interação na tela, como sugerido pelo guia de interface gráfica de iOS (APPLE, 2014b).

4. **Customização de cores e emoções** - Não somente aspectos físicos variam de usuário para usuário. O meio em que cresceram, sua cultura e costumes também podem impactar na utilização do aplicativo. Por exemplo, a cor vermelha pode simbolizar o amor, mas também pode simbolizar a raiva. Portanto, o aplicativo deve permitir a customização das cores atreladas às emoções contidas nas entradas do diário, permitindo uma associação mais adequada para seu contexto.

5. **Orientação** - Como indicado por Gazzaley (2012), as pessoas estão propensas a perder a atenção em seu aplicativo devido a fatores externos. Com a finalidade de manter o usuário informado sobre a posição no aplicativo que está atualmente, este deve adotar uma estrutura hierárquica de navegação de telas. Desta maneira, o aplicativo sempre informa se é possível voltar na estrutura e qual o nome da tela que o usuário está vendo. Estas duas informações permitem que o usuário possa se localizar, mesmo que tenha sua atenção interrompida.

6. **Internacionalização** - Um aplicativo normalmente é lançado em todo o mundo. Com o propósito de permitir que usuários de outros países entendam e utilizem o aplicativo, ele deve ser internacionalizado para o inglês (que é considerada a língua padrão para se comunicar quando fora de seu país nativo). Como o aplicativo fruto deste trabalho de conclusão de curso foi desenvolvido no Brasil, a implementação da internacionalização na língua mãe do país, português brasileiro, também deve estar presente.

7. **Compartilhamento** - A Internet é praticamente indispensável e o Brasil é um país que compõe um grande volume de internautas. Segundo (EMARKETER, 2014), no primeiro trimestre deste ano, 89 milhões de brasileiros acessaram o Facebook mensalmente e 59 milhões acessaram diariamente. Face estes números, dar ao usuário a opção de compartilhar fotos do seu diário com uma breve descrição também é um ponto desejável e que deve estar presente no aplicativo.

Estes tópicos foram tratados como requisitos funcionais e não funcionais do aplicativo desenvolvido. A maneira como cada um destes tópicos foram abordados está presente em detalhes no Capítulo 5.

3.2 Restrições do sistema proposto

O aplicativo não permitirá que uma mesma foto seja usada para mais de uma entrada no diário devido a restrições do próprio sistema iOS. O sistema operacional também não permite remover uma foto do álbum iOS através do aplicativo, somente a referência àquela foto. Para que as fotos sejam removidas completamente do dispositivo, isso deve ser feito através da aplicação de fotos nativa no sistema iOS.

3.3 Recursos necessários para a execução

Para o desenvolvimento do projeto são necessários recursos tanto de hardware quanto de software, os quais são descritos nas próximas subseções.

3.3.1 Descrição do hardware ideal

O aplicativo Journpic foi desenvolvido para sistema iOS. Existem vários modelos de dispositivos móveis com este sistema, mas eles se concentram em duas categorias: iPad e iPhone. A seguir é descrito o hardware ideal para executar o aplicativo em cada uma destas categorias.

Para os dispositivos iPad

- Modelo - iPad Air
- Tipo de tela - Retina
- Dimensão da tela - 9.7 polegadas
- Processador - Chip A7
- Capacidade - 32GB

Para os dispositivos iPhone

- Modelo - iPhone 6 Plus
- Tipo de tela - Retina HD
- Dimensão da tela - 5.5 polegadas
- Processador - Chip A8
- Capacidade - 128GB

3.3.2 Descrição do hardware mínimo

Os modelos mais antigos de iPhone e iPad que conseguem executar o aplicativo Journpic são descritos a seguir:

Para os dispositivos iPad

- Modelo - iPad Mini
- Tipo de tela - Convencional
- Dimensão da tela - 7.9 polegadas
- Processador - Chip A5
- Capacidade - 16GB

Para os dispositivos iPhone

- Modelo - iPhone 5C
- Tipo de tela - Retina
- Dimensão da tela - 4 polegadas
- Processador - Chip A6
- Capacidade - 8GB

3.3.3 Descrição do software

Para produção e execução do aplicativo Journpic foram necessários vários softwares. A descrição detalhada de cada um deles pode ser vista a seguir:

- OS X Yosemite - Sistema operacional do MacBook. Ele é necessário para executar a ambiente de desenvolvimento.
- iOS 8.1 - Sistema operacional dos dispositivos móveis. O dispositivo mais antigo que tem suporte a esta versão do sistema é o iPhone 4S.
- XCode 6 - Ambiente de desenvolvimento que conta com ferramentas de análise de vazamento e consumo de memória, carga de processamento e permi e a emulação dos vários dispositivos iOS.
- Astah Community - Ferramenta gratuita utilizada para modelagem de artefatos utilizados pela Engenharia de Software.
- Mozzila Firefox - Navegador de Internet. Referências para soluções de problemas de código e demais tópicos que pudessem gerar dúvidas podem ser pesquisados mais rapidamente utilizando um navegador.

- POP - Aplicativo multiplataforma destinado à criação de protótipos de interface gráfica.

3.3.4 Rede

Os sistemas operacionais iOS já possuem uma configuração de rede pré-determinada, podendo se conectar à Internet através de rede wifi ou rede de telefonia móvel. Independente da maneira utilizada, o aplicativo não tem permissão para alterar configurações de rede do dispositivo. Portanto, não existiu necessidade para a criação de uma rede para o aplicativo.

3.3.5 Banco de dados

Para o desenvolvimento deste aplicativo foi utilizado o banco de dados SQLite, nativo dos sistema iOS.

4 Desenvolvimento do Sistema

Este capítulo aborda os planos adotados para a execução deste trabalho de conclusão de curso, contemplando tanto o aspecto gerencial quanto o de implementação. Os aspectos gerenciais foram abordados com base nos conhecimentos de Gestão de Portfólio de Projeto, Requisitos de Software e Métodos de Desenvolvimento de Software. Já os aspectos de implementação consideraram conhecimentos de Orientação à Objetos, Estrutura de Dados e Algoritmos, Padrões de Projeto de Software, Gestão da Configuração de Software, Manutenção e Evolução de Software, entre outros.

4.1 Planejamento do Projeto

4.1.1 Plano do processo de desenvolvimento

A seguir estão descritos todos os procedimentos adotados no processo de desenvolvimento do aplicativo proposto neste projeto. O planejamento e desenvolvimento de todo este projeto foi realizado utilizando-se de conhecimentos de Engenharia de Software. Estes conhecimentos estão contidos no guia SWEBOK (2007).

4.1.1.1 Ciclo de vida do projeto

Para o desenvolvimento do aplicativo Journpic foi adotada uma adaptação da abordagem interativa e incremental, chamada Scrum. Esta abordagem é bastante objetiva, apresenta metas claras, fornece métodos para a definição de planejamento, forma de trabalho da equipe, principais papéis de pessoas envolvidas, flexibilidade, cooperação e comprometimento. A metodologia ocorre através do desenvolvimento e interação de novas versões do software contendo novas funcionalidades e/ou melhorias sobre as funções já existentes. Fonseca (2009) afirma que a equipe desenvolve de forma interativa realizando todas as fases de desenvolvimento para cada *Select Backlog* (“requisito”) antes de avançar para a próxima *Sprint*, incrementando sempre novas funcionalidades e/ou melhorias.

A escolha desta metodologia de desenvolvimento se deve ao caráter ágil cenário, onde há tempo limitado para a realização do projeto, além da avaliação do produto em desenvolvimento que deve ser realizada em cada iteração. Esta escolha foi tomada ao comparar o SCRUM ao tradicional *Rational Unified Process* (RUP). Este último é muito utilizado em grandes equipes, grandes projetos e bastante comum em fábricas de software, no qual prevalece a documentação, o controle do escopo, requisitos, qualidade e papéis

bem definidos (KRUCHTEN, 2004). Como o aplicativo proposto é de pequeno porte e possui maleabilidade do escopo, foi escolhido o SCRUM.

A metodologia escolhida divide o trabalho em interações de duas a quatro semanas no máximo definidas como *Sprint* que é a unidade desenvolvimento (FONSECA, 2009). A cada *Sprint* realizada o ciclo de desenvolvimento do software é aplicado. No processo de desenvolvimento do aplicativo em questão foi necessária, contudo, apenas uma semana para que cada iteração fosse desenvolvida devido ao domínio do desenvolvedor nas tecnologias e linguagem de programação utilizadas. Nas iterações ocorridas durante o desenvolvimento do projeto houve a necessidade de cumprir as seguintes fases do modelo escolhido:

- Reunião de planejamento da *Sprint* - reunião onde foram definidas as funcionalidades que serão tratadas na iteração de acordo com a prioridade do trabalho.
- Reunião diária - reuniões diárias realizadas pela equipe de desenvolvimento para um acompanhamento sobre o progresso da implementação, as dificuldades e o que seria feito até o próximo encontro de maneira objetiva, clara e rápida.
- Reunião de revisão da *Sprint* - reuniões onde verificavam-se se o objetivo foi atingido, onde realizaram-se também a validação da entrega. Essas reuniões só aconteceram ao final de cada *Sprint* para apresentação das funcionalidades implementadas.
- Retrospectiva da *Sprint* - retrospectiva sobre a última *Sprint*. Esta etapa foi realizada após a Reunião de revisão da *Sprint*, apenas pela equipe de desenvolvimento para uma avaliação da *Sprint* sob a visão de processo, produto ou equipe, além de descobrir os acertos e erros com intuito de realizar uma melhoria no processo de desenvolvimento.

4.1.1.2 Testes de Software

Durante o desenvolvimento do aplicativo foram aplicados testes funcionais, os quais avaliam o comportamento da aplicação. Dados de entrada são fornecidos, o teste é executado e o resultado obtido é comparado com o resultado esperado. Para planejar testes, Bertolino (2007) considerou algumas questões. Estas questões (Tabela 2) foram tomadas como modelo para planejar os testes.

Desconsiderando os testes intrínsecos ao processo de codificação, que normalmente serve para validar o trecho de código escrito, este projeto contou com testes de funcionamento dos casos de uso levantados a partir dos requisitos apresentados anteriormente no Capítulo 3. A Figura 24 contida Apêndice A apresenta os casos de uso que podem ser executados pelo usuário e que foram priorizadas durante os testes funcionais.

Questão	Conceito
Porque?	Motivo do teste
Como?	Seleção do teste
Quanto?	Adequação do teste
Qual?	Foco do teste
Onde?	Ambiente do teste
Quando?	Momento do teste

Tabela 2: Questões para planejamento de testes

4.1.2 Plano de acompanhamento

A seguir estão listados todos os procedimentos adotados para o acompanhamento e controle sobre o projeto desenvolvido bem como suas respectivas descrições.

4.1.2.1 Marcos e pontos de controle

Os marcos foram programados para acontecer de acordo com a necessidade ao longo do projeto, estas verificações foram definidas pelo tutor Phelippe Amorim e o aluno autor do projeto. Estes pontos de controle foram realizados em intervalos de aproximadamente 15 dias pelo tutor, que fez a verificação do cumprimento de atividades do período com qualidade mínima e o respeito ao cronograma proposto.

4.1.2.2 Análise e gerência de riscos

A Análise e Gerência de Riscos visa prevenir problemas que possam surgir no decorrer da implementação do software, podendo influenciar tanto na qualidade quanto no cronograma do projeto. A seguir estão descritos os possíveis riscos para a implementação do aplicativo Journpic que foi proposto por este projeto:

- Atraso na entrega do projeto - Concentrar os esforços por parte do aluno autor, aumentando a carga de trabalho voltada para o desenvolvimento do projeto.
- Erro no levantamento e especificação dos requisitos - Revisão dos requisitos pelo orientador.
- Implementação de funcionalidade não condizente ao requisito - Adequação da funcionalidade perante a especificação do caso de uso e revisão do caso de uso pelos orientador.
- Falta de conhecimento na atividade a ser executada - Estudar os atributos e assuntos necessários para obtenção do conhecimento necessário que auxiliarão na realização da atividade proposta.

- Arquivos de código fonte corrompidos - Utilizar um repositório remoto que possua suporte ao versionamento.

4.1.3 Cronograma

O cronograma a seguir mostra as principais atividades desenvolvidas no decorrer do semestre acadêmico da Universidade de Brasília.

AGOSTO

Dia 13 - Concepção inicial da proposta.

Dia 15 - Validação da proposta com o professor André Barros.

Dia 18 - Levantamento de informações sobre trabalhos correlatos.

Dia 25 - Levantamento inicial de requisitos.

SETEMBRO

Dia 11 - Data limite para inscrição no Trabalho de Conclusão de Curso 2.

Dia 15 - Definição das regras de negócio.

Dia 22 - Análise das regras de negócio definidas.

Dia 26 - Documentação inicial do projeto final.

Dia 29 - Início da primeira *Sprint*.

OUTUBRO

Dia 06 - Início da segunda *Sprint*.

Dia 13 - Início da penúltima *Sprint*.

Dia 20 - Início da última *Sprint*.

Dia 27 - Complementação da documentação do projeto final.

NOVEMBRO

Dia 03 - Atualização da documentação do projeto com conclusão e resultados.

Dia 13 - Homologação do projeto.

Dia 18 - Entrega do projeto.

Dia 26 - Apresentação do projeto.

4.2 Tecnologias

Para o desenvolvimento deste projeto foram tomadas uma série de medidas aprendidas durante o curso de Engenharia de Software para garantir a qualidade do produto final e minimizar os riscos envolvidos no desenvolvimento. Uma explicação mais detalhada sobre as tecnologias e padrões utilizados pode ser vista neste capítulo.

4.2.1 Métodos de desenvolvimento e ferramentas CASE

O projeto foi desenvolvido utilizando o paradigma de análise Orientado a Objetos (OO), pelos benefícios que ele traz referente ao reaproveitamento e a qualidade de software (BASILI; BRIAND; MELO, 1996). Foi utilizada também a Unified Modelling Language (UML) para a análise e modelagem do aplicativo, fruto deste projeto.

Na Tabela 3 segue uma descrição das ferramentas de desenvolvimento e Computer-Aided Software Engineering (CASE) que foram utilizadas neste projeto.

Ferramenta	**Descrição**
XCode 6	Integrated Development Enviroment (IDE)
Astah Community 6.9	Modelagem de artefatos UML
SQlite 3	Banco de dados relacional
POP 2.0.14	Prototipagem

Tabela 3: Ferramentas de desenvolvimento

A escolha destas tecnologias também levou em consideração o gasto necessário para obter os respectivos softwares com o propósito de minimizar possíveis gastos. Todas as ferramentas de software utilizadas são gratuitas e, portanto, não acarretaram custos ao projeto.

4.2.2 Linguagens de programação

A linguagem de programação do aplicativo foi o Objective-C, pois é a linguagem nativa suportada por sistemas desenvolvidos para iOS. Esta linguagem se assemelha ao C, é orientada a objetos e possui uma sintaxe de envio de mensagens semelhante a linguagem Small Talk.

O banco de dados do aplicativo foi construído em SQLite pelos seguintes motivos: é o banco de dados nativo das aplicações iOS; os requisitos do aplicativo não apontaram a necessidade de um banco de dados mais robusto. Como a a tecnologia de banco de dados padrão já era suficiente para suportar o aplicativo, não houve necessidade de buscar outra alternativa.

Na Tabela 4 segue uma descrição das linguagens de programação que foram utilizadas neste projeto.

Tipo de linguagem	**Descrição**
Objective-C	Linguagem reflexiva orientada a objetos
SQL	Linguagem de consulta estruturada

Tabela 4: Linguagens de programação

4.2.3 Ambiente de hardware para o desenvolvimento

Levando em consideração o hardware mínimo e recomendado para a execução do aplicativo, foram utilizados os seguintes recursos de hardware durante o desenvolvimento deste projeto:

1. Notebook

- Modelo - MacBook Pro 13"
- Processador - Intel Core i5
- Velocidade do processador - 2.6GHz
- Memória - 8GB
- Capacidade - 512GB
- Sistema operacional - OS X Yosemite

2. Tablet

- Modelo - iPad Air
- Processador - Chip A7
- Tamanho da tela - 9.7 polegadas
- Capacidade - 32GB
- Sistema operacional - iOS 8.1

3. Smartphone

- Modelo - iPhone 5C
- Processador - Chip A6
- Tamanho da tela - 4 polegadas
- Capacidade - 8GB
- Sistema operacional - iOS 8.1

4.2.4 Controle de Configuração

O controle de configuração de software se preocupa com a gestão de mudanças durante o ciclo de vida do software. Abrange o processo de determinação sobre quais mudanças fazer, a autoridade para aprovar certas mudanças, o apoio para a implementação dessas mudanças, e o conceito de desvios formais de requisitos do projeto, bem como isenção deles. Informações obtidas a partir dessas atividades é útil em medir o tráfego de mudança e ruptura, e aspectos de retrabalho (SWEBOK, 2007).

Como apontando na análise de riscos contida no plano de acompanhamento (Seção 4.1.2), existe a importância tanto em evitar que arquivos de código fonte sejam corrompidos, quanto em voltar a versões anteriores do código, fato constatado durante o curso de Engenharia de Software. Portanto, para o desenvolvimento deste projeto foi criado um repositório remoto privado no GitHub (`https://github.com/`) para possibilitar um desenvolvimento distribuído e com suporte ao controle de versões.

4.2.5 Modelo de Arquitetura

Em seu sentido estrito, uma arquitetura de software é "uma descrição dos subsistemas e componentes de um sistema de software e as relações entre eles." (BUSCHMANN et al., 2013). Arquitetura tenta, assim, definir a estrutura interna - de acordo com o Dicionário de Inglês Oxford, "a maneira em que algo é construído ou organizada" - do software resultante. Em meados da década de 1990, no entanto, a arquitetura de software começou a emergir como uma disciplina mais ampla, envolvendo o estudo de estruturas e arquiteturas de software de uma forma mais genérica (SHAW; GARLAN, 1996). Isso deu origem a uma série de ideias interessantes sobre design de software em diferentes níveis de abstração. Alguns desses conceitos podem ser úteis durante o projeto arquitetônico (por exemplo, o estilo arquitetônico) de software específico, bem como durante a sua concepção detalhada (por exemplo, padrões de projeto de nível mais baixo). Mas eles podem também ser úteis para a concepção de sistemas genéricos, levando à criação de famílias de programas (também conhecidas como linhas de produtos). Curiosamente, a maioria destes conceitos pode ser vista como uma tentativa de descrever, e, assim, reutilizar, design conhecimento genérico (SWEBOK, 2007).

Pressman (2011) cita que uma arquitetura em três camadas (separada em interface, navegação e comportamento) simplifica a implementação e aumenta a reutilização. A arquitetura Model-View-Controller (MVC) é uma das arquiteturas que seguem este conceito. O *model* contém a lógica de processamento, objetos de conteúdo, acessos às fontes de dados e fontes externas, e funcionalidades de processamento específica para a aplicação. A *view* abrange todas as funcionalidades da interface, possibilitando a apresentação do conteúdo, da lógica de processamento, dos acessos às fontes, e toda a funcionalidade de processamento requirida pelo usuário. A *controller* atua como uma ponte, gerenciando

o acesso entre o *model* e a *view* e coordenando o fluxo de dados entre eles.

A Figura 2 apresenta um esquemático desta arquitetura, mostrando a relação entre as três camadas que a compõem e como elas se relacionam entre si.

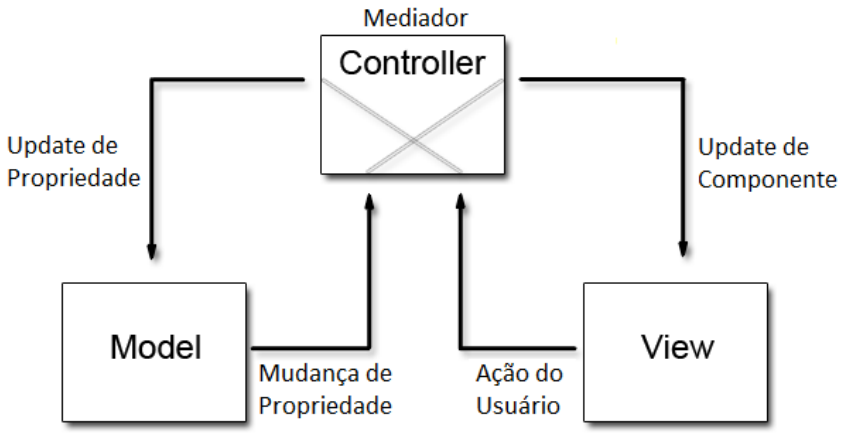


Figura 2: Modelo Model-View-Controller (MVC)

4.3 Implementação

Este projeto foi realizado dentro do programa BEPiD, em parceria com a Universidade Católica de Brasília. A implementação do banco de dados e integração das galerias de fotos com o dispositivo móvel foram desenvolvidas dentro do quadro do programa. Já as demais etapas, descritas ao longo deste capítulo, foram desenvolvidas dentro deste trabalho de conclusão de curso.

4.3.1 Model-View-Controller

Como proposto pela própria Apple (2014a), o padrão de projeto de software adotado na construção deste aplicativo foi o Model-View-Controller (Figura 2), por ser simples, robusto, orientado a objetos e por ter a maior compatibilidade ao seguir o ciclo de vida dos aplicativos desenvolvidos para a plataforma iOS, como apresentado no Anexo A. Os tipos de classe criados se dividiram, portanto, nos grupos de modelo, controladora e visão.

4.3.1.1 Classes de Modelo

As classes modelo foram as primeiras a serem criadas. O aplicativo Journpic possui três entidades de modelo: Photo, Tag e Emotion. Uma foto pode ter uma tag, mas uma mesma tag pode estar presente em várias fotos. Uma foto poder ter uma emoção atribuída

a ela, mas uma mesma emoção pode aparecer em várias fotos. A representação visual disto está apresentado na Figura 3.

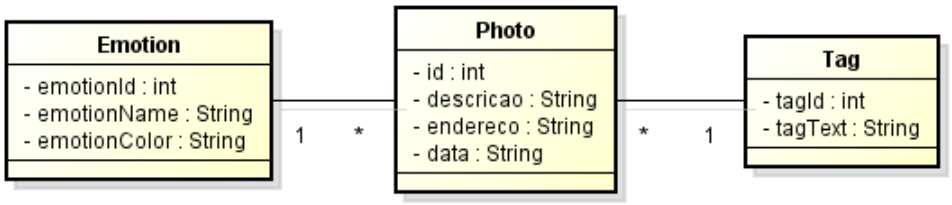


Figura 3: Relação entre as classes de modelo

Com a implementação destas classes concluídas, foi possível proceder com o próximo passo: A criação das classes de controladora.

4.3.1.2 Classes de Controladora

A linguagem de programação utilizada para desenvolver o Journpic foi o Objective-C. Nesta linguagem cada classe possui dois arquivos: o primeiro é referente à interface (.h) e o segundo é utilizado para a implementação dos métodos presentes na interface citada (.m), além de outros métodos auxiliares. A disposição desta maneira provê o encapsulamento das informações.

As controladoras compõem a camada intermediária deste software, estando localizadas entre as classes de modelo e as classes de visão. O papel da controladora é fazer o tratamento dos dados inseridos pelo usuário na camada de visão, disparando mensagens à outras controladoras e acessando as camadas inferiores, como as classes modelo. As controladoras, portanto, são responsáveis pela maior parte do funcionamento do software.

O Algoritmo 1 exibe um exemplo de controladora e alguns pontos devem ser ressaltados.

- Os nomes de classes tendem a começar com as iniciais do nome do desenvolvedor ou pacote. Por exemplo, as iniciais JP significam que esta classe pertence ao Journpic, já UI se refere ao kit de User Interface.

- O termo IBOutlet significa que a @property em questão pode ser conectada diretamente a um elemento de interface gráfica, como botão e campo de texto. Na linha 8 é criada uma property para o campo textual de descrição desta tela na interface gráfica (View).

- Já o IBAction tem a finalidade semelhante ao IBOutlet. O IBAction determina um método que poderá ser conectado a um elemento de interface gráfica, atribuindo a este elemento uma ação.

JPPhotoInfoViewController.h

```
#import <UIKit/UIKit.h>

@protocol JPPopViewDelegate <NSObject>
- (void)sendDataBackToFirstController:(BOOL) isPopEnabled;
@end

@interface JPPhotoInfoViewController : UIViewController <UITextFieldDelegate,
→  UICollectionViewDataSource, UICollectionViewDelegate>
@property (weak, nonatomic) IBOutlet UITextField *descriptionField;
@property (weak, nonatomic) IBOutlet UITextField *tagField;
@property (weak, nonatomic) IBOutlet UICollectionView *collectionView;
@property (weak, nonatomic) IBOutlet UILabel *descriptionLabel;
@property (weak, nonatomic) IBOutlet UILabel *tagLabel;
@property (weak, nonatomic) IBOutlet UIButton *dateInThePastButton;

@property (nonatomic, strong) id <JPPopViewDelegate> popDelegateObject;

@property (nonatomic) NSMutableArray *photoEmotion;
@property (nonatomic) NSMutableDictionary *photoInfo;

- (IBAction)dateInThePastModal:(id)sender;

- (void)saveDataToDataBase;
```

Algoritmo 1: Interface da controladora JPPhotoInfoViewController.

Com o uso destes conectores, o código da controladora deixa de estar atrelado diretamente a um único elemento, podendo ser reaproveitado. Isto diminui o acoplamento entre as classes e aumenta a coesão de cada um dos métodos, que é a intenção do padrão MVC.

4.3.1.3 Views

A próxima camada é a de visão. O ambiente de desenvolvimento XCode possibilita a criação de views (telas) através de uma ferramenta gráfica onde o desenvolvedor escolhe quais itens de interface deseja e junta-os como achar mais adequado, desde que siga a ordem hierárquica dos elementos.

Neste ambiente, os arquivos gerados para armazenar o resultado da junção de coleções, paletas e objetos de interface têm o formato NeXT Interface Builder (.xib) (Fig. 5).

A Figura 5 mostra o XIB associado à classe JPMainCollectionViewController (presente no Apêndice). É possível notar que esta XIB possui uma *Navigation Controller*, e que em sua *Navigation Bar* contém um *Bar Button Item* à esquerda, representando o menu, e outro *Bar Button Item* à direita, como botão de acesso à câmera. A *Navigation Controller* citada engloba uma *Collection View Controller*, a qual, por sua vez, engloba uma *Collection View*. A *Collection View* possui uma *Collection View Cell* de protótipo e nela há uma *Image View*. Esta *Image View* simboliza a foto associada à entrada do diário e ocupa todo o tamanho da célula de protótipo, sendo que dentro dela exite uma *View* menor, no canto inferior direito, que tem duas *Label* dispostas uma sobre a outra: a *Label*

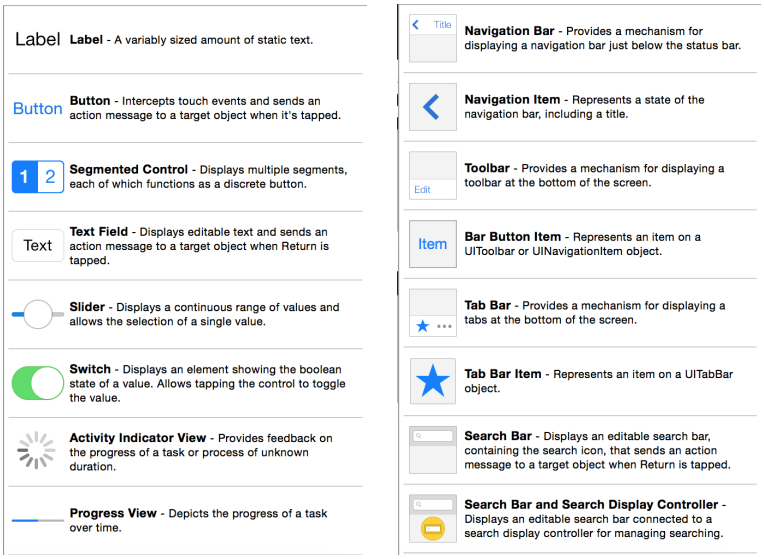


Figura 4: Alguns elementos de interface

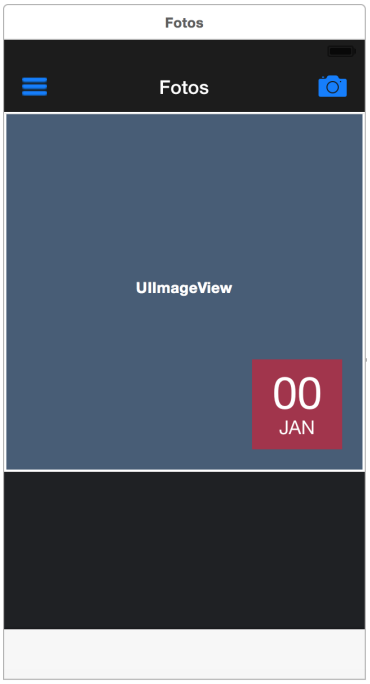


Figura 5: Arquivo XIB referente à tela principal do aplicativo.

superior se refere ao dia do mês da entrada do diário em questão, e a inferior ao mês.

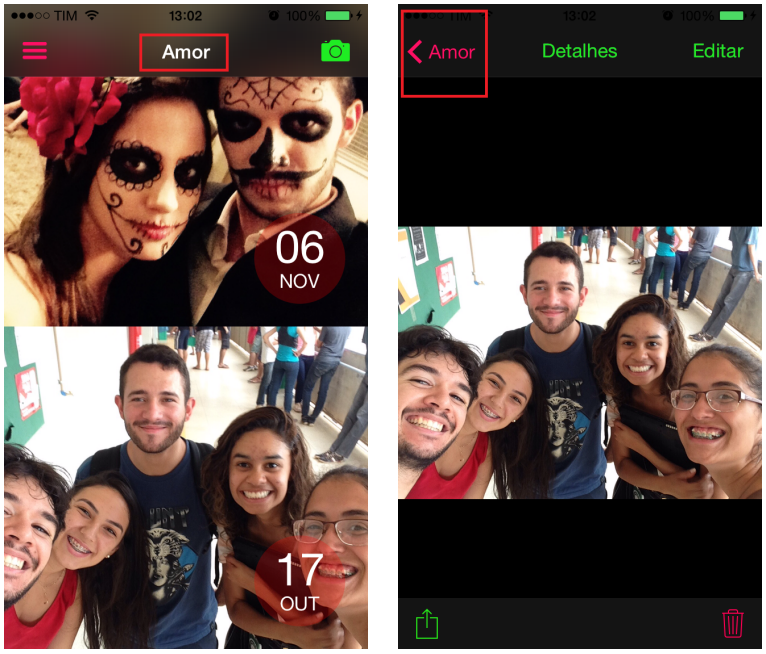
4.3.2 Adaptabilidade

Dada a explicação sobre as views, é importante ressaltar que este projeto teve como um de seus requisitos contornar o problema da adaptabilidade, já que existem dispositivos móveis que têm o tamanho de tela variando de 3,5 polegadas a 9,7 polegadas que rodam o sistema operacional iOS. O usuário deseja que um mesmo aplicativo seja executado adequadamente independente do dispositivo em questão, então para solucionar

este problema foram utilizadas *Constraints* (restrições), que funcionam como amarras definindo a posição que cada elemento deve ficar na tela. Por exemplo, a *Image View* apresentada na Figura 5 possui quatro *constraints*, uma de cada lado da sua borda, definindo que a distância entre a borda superior da célula e a borda superior da *Image View* é de zero pontos, que a borda esquerda da célula e a borda esquerda da *Image View* também é zero, e assim por diante. Desta maneira, é garantido que a imagem contida na célula sempre ocupará toda sua área independente do tamanho da célula em si, que varia de acordo com o tamanho do dispositivo móvel.

4.3.3 Estrutura de navegação

Os requisitos do sistema também explicitaram a necessidade de tratar a perda de atenção. Através do uso de *Navigation Controllers*, um tipo de controladora destinada a aplicativos com fluxos hierárquicos de tela, foi possível tratar o problema de desorientação devido à perda de atenção, como descrito nos objetivos específicos deste projeto. A *Navigation Controller* gera uma barra de navegação na parte superior da tela, apresentando o título da tela atual e a opção de voltar um nível na hierarquia de telas. O aplicativo Journpic possui uma navegação de nível geral (nível mais alto) até detalhes (nível mais baixo) a medida que descemos na hierarquia, então esse tipo de controladora atende perfeitamente a necessidade. Este padrão está definido no Guia de Interface Humana da Apple (APPLE, 2014b).



(a) Navegação em nível geral (b) Detalhes do item

Figura 6: Estrutura de navegação hierárquica

A Figura 6a apresenta a navegação em nível superior por meio de uma *Navigation View Controller* de título "Amor", onde o usuário pode tocar uma das células da *Collection*

View e navegar um nível adentro na hierarquia de telas, acessando a tela de detalhes (Figura 6b). Neste nível de hierarquia o *Bar Button Item* à esquerda na *Navigation Bar* é substituído pelo botão de voltar com o título da tela acima na hierarquia.

A IDE também permite a criação de storyboard, que é uma ferramenta gráfica para definir fluxo de views do aplicativo. O storyboard geral, contendo todo o fluxo de telas do aplicativo Journpic pode está representado na Figura 7.

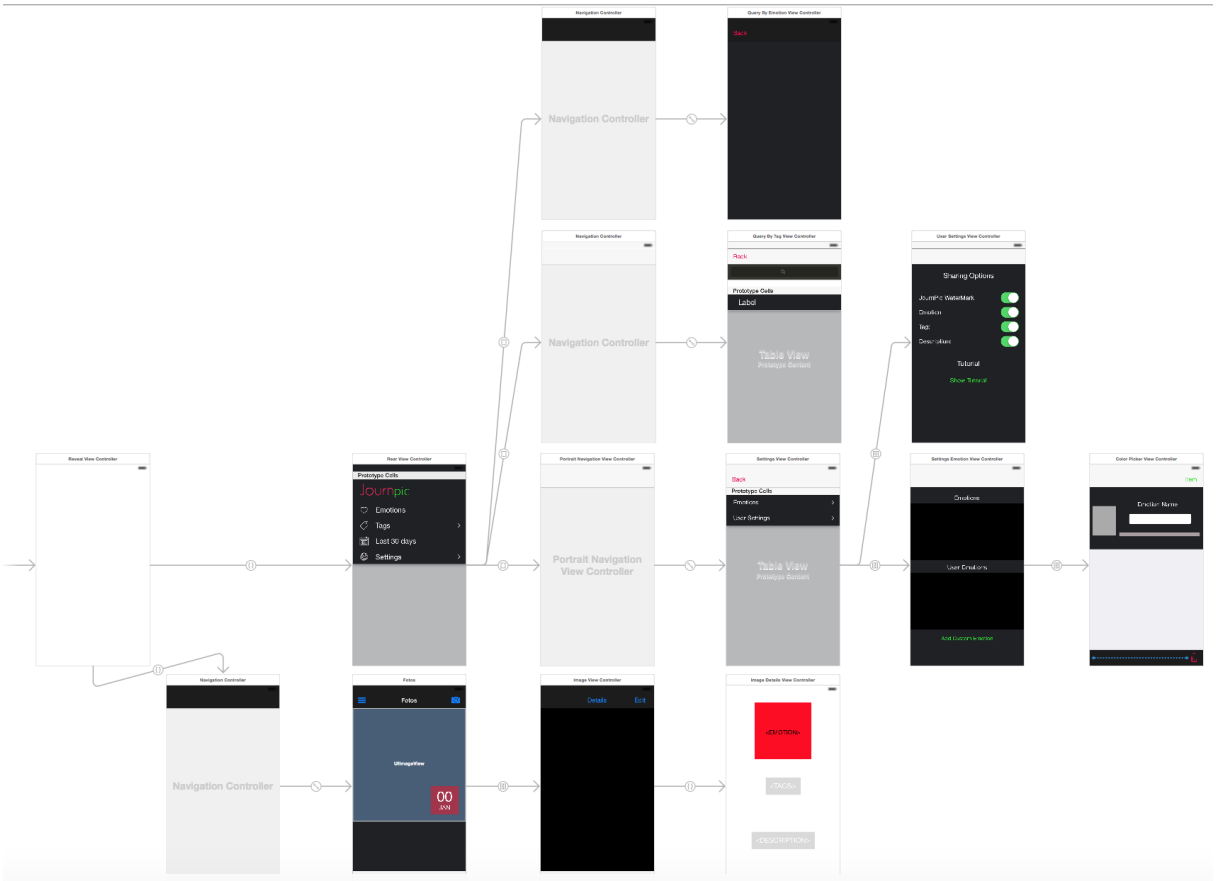


Figura 7: Estrutura de navegabilidade do aplicativo Journpic

4.3.4 Funcionalidade de foto

No sistema iOS, um objeto do tipo *ALAsset* representa uma foto ou vídeo gerenciado pelo aplicativo de fotos nativo do sistema. Como o sistema não permite acesso direto às fotos contidas dentro do disco rígido do dispositivo móvel, a utilização deste tipo de objeto é inevitável para aplicativos que desejam acessar fotos e vídeos ou guardá-las no álbum do próprio aparelho.

O aplicativo precisou atrelar as informações de tag e emoção das fotos. Para que o aplicativo conseguisse acessar o álbum do dispositivo com o intuito de gravar informações foi necessário autorizá-lo. Para isto, foi preciso implementar um método para autorização como visto no Algoritmo 2.

```
- (void)autorizacao {

    ALAuthorizationStatus status = [ALAssetsLibrary authorizationStatus];

    if( status == ALAuthorizationStatusAuthorized ){
        // authorized
        NSLog(@"authorized");
        [self initAlbum];

    } else if ( status == ALAuthorizationStatusDenied || status ==
    ALAuthorizationStatusRestricted ) {
        // denied or restricted, may ask user grant for photo permission in
    Setting->Privacy->Photos
        NSLog(@"denied or restricted, Setting->Privacy->Photos");
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:NSLocalizedString
    (@"PermissionDenied.message", nil) message:NSLocalizedString
    (@"PermissionDeniedText.message", nil)delegate:nil cancelButtonTitle:NSLocalizedString
    (@"Close", nil)  otherButtonTitles:nil, nil];
        [alert show];

    } else if ( status == ALAuthorizationStatusNotDetermined ){
        //ALAssetsGroupLibrary, ALAssetsGroupSavedPhotos
        // ask for permission
        [_assetLibrary enumerateGroupsWithTypes:ALAssetsGroupLibrary
    usingBlock:^(ALAssetsGroup *group, BOOL *stop) {

            if (group == nil) {
                // user select ok
                NSLog(@"group nil");
            }
            *stop = YES;
        } failureBlock:^(NSError *error) {
            NSLog(@"error enumerating AssetLibrary groups %@\n", error);
        }];

    }
}
```

Algoritmo 2: Método de autorização de acesso ao álbum do iOS

Com este passo concluído, foi possível gravar imagens no álbum. Ao tocar o botão com o ícone de uma máquina fotográfica no canto superior direito na tela principal do aplicativo, o usuário escolhe entre escolher uma foto ou tirar uma foto (Figura 8). O método chamado quando a opção de escolher uma foto é selecionada pode ser visto no Algoritmo 3.

Inserindo as informações de emoção e tag na foto selecionada, o usuário pode emfim salvar sua nova entrada no diário. O aplicativo adquire as informações referentes ao *Asset* da imagem, juntamente com a emoção, tag e descrição, caso exista, e insere nas tabelas correspondentes no banco de dados. O número de passos necessários para garantir que tudo funcione em harmonia e com a devida qualidade, tratamentos de erros e encapsulamento é bastante extenso, e por isso será apresentado apenas o método do Algoritmo 4, o qual é responsável por adicionar a foto à biblioteca do dispositivo móvel e chamar o método que irá salvar os dados no banco. Todos os passos podem ser rastreados conferindo o código fonte completo do aplicativo no apêndice.

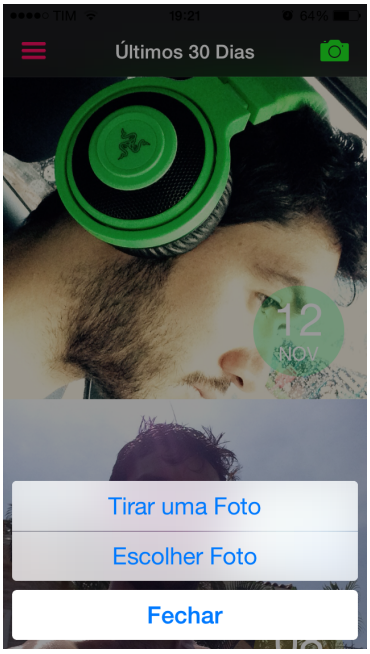


Figura 8: O usuário deve escolher entre escolher uma foto ou tirar uma foto.

```
1  -(void)choosePhoto {
2
3      UIImagePickerController *mediaUI = [[UIImagePickerController alloc] init];
4      mediaUI.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;
5      //Display saved pictures and movies from the camara roll
   →   mediaUI.mediaTypes = [UIImagePickerController availableMediaTypesForSourceType:
6  →   UIImagePickerControllerSourceTypePhotoLibrary];
   →   mediaUI.mediaTypes = [[NSArray alloc] initWithObjects: (NSString *) kUTTypeImage,
7  →   nil];
8
9      mediaUI.allowsEditing = NO;
10     mediaUI.delegate = self;
11
12         [[NSOperationQueue mainQueue] addOperationWithBlock:^{
13             // Place image picker on the screen
14             [self presentViewController:mediaUI animated:YES completion:nil];
15         }];
16 }
```

Algoritmo 3: Método para selecionar uma foto do álbum

4.3.5 Internacionalização

Aplicações desenvolvidas no ambiente de desenvolvimento XCode são bastante simples de ser internacionalizadas. Para a internacionalização do aplicativo Journpic foi necessário que todas as mensagens, avisos, títulos e afins fossem escritos em um arquivo de texto, um para cada idioma. Estes arquivos possuem uma extensão informando qual é sua língua, e em tempo de execução o aplicativo carrega os textos adequados à linguagem que o sistema iOS está operando no momento. Caso a linguagem do sistema operacional não esteja em nenhuma das linguagens pré definidas, o aplicativo é executado na linguagem padrão (no caso, o idioma inglês). Como os arquivos são carregados por sua extensão, é possível internacionalizar imagens (usar imagens de bandeiras distintas, por exemplo) ao

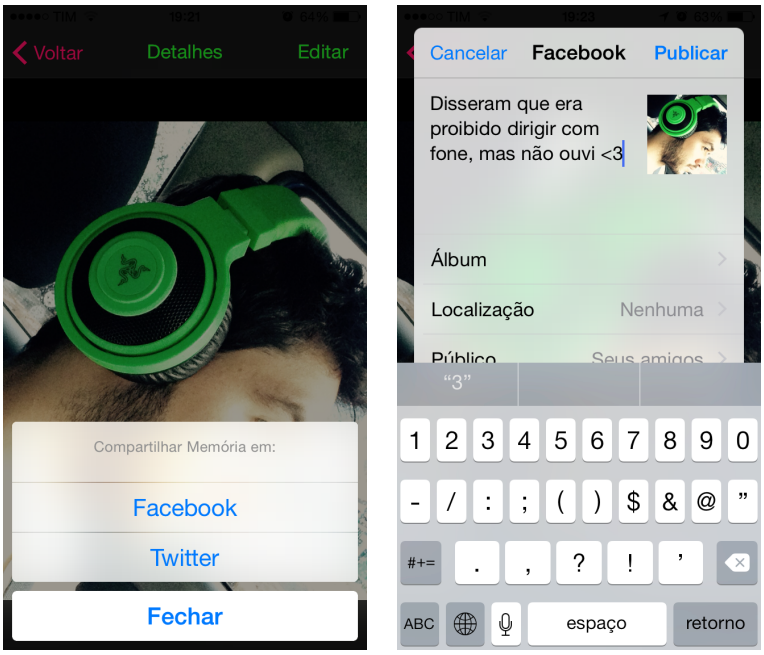
```
#pragma mark - Salvar Fotos no album JournPic [CAMERA]

- (void)salvarDaCamera:(UIImage *)imagem informacoes:(NSDictionary *)info
{
    CGImageRef img = [imagem CGImage];
    [_assetLibrary writeImageToSavedPhotosAlbum:img
    metadata:[info objectForKey:UIImagePickerControllerMediaMetadata]
    completionBlock:^(NSURL* assetURL, NSError* error) {
        if (error.code == 0) {
        NSLog(@"saved image completed:\nurl: %@", assetURL);
        // try to get the asset
        [_assetLibrary assetForURL:assetURL
        resultBlock:^(ALAsset *asset) {
            // assign the photo to the album
            [self salvarNoBanco:assetURL];
            [_groupToAddTo addAsset:asset];
            [_photoAssets addObject:asset];
            [_photoAssetsURL addObject:assetURL];
            [self atualizarCollectionView];
        }
        failureBlock:^(NSError* error) {
            NSLog(@"failed to retrieve image asset:\nError: %@ ", [error
→   localizedDescription]);
        }];
        }
        else {
            NSLog(@"saved image failed.\nerror code %li\n%@", (long)error.code, [error
→   localizedDescription]);
        }
    }];
}
```

Algoritmo 4: Método para salvar uma foto da câmera no álbum

informar o nome do arquivo de imagem a ser carregado, assim como são carregados os demais textos.

4.3.6 Funcionalidade de compartilhamento

Um dos requisitos deste projeto prevê que a aplicação desenvolvida tenha a funcionalidade de publicar em redes sociais. Para sanar este requisito, foram implementadas as opções de compartilhar entradas do diário tanto no *Facebook* quanto no *Twitter*. Esta opção está disponível na parte inferior esquerda da tela de visualização de imagem e fornece ao usuário as duas opções de publicação através de uma *UIActionSheet* (Figura 9a). Ao selecionar uma delas, os métodos de sua respectiva Interface de Programação de Aplicativos (API) é chamada para que o usuário preencha os campos específicos, como texto, público e localização e por fim pode publicar (Figura 9b). O Algoritmo 5 mostra o método utilizado para a publicação no *Facebook*. A Figura 10 mostra a foto publicada.



(a) Opções de publicação (b) Campos da publicação

Figura 9: Publicando no facebook

Figura 10: Foto publicada no Facebook

```
#pragma mark - Compartilhar Facebook

- (void)compartilharFacebook:(UIImage *)imagem eComentario:(NSString *)comentario
{
    if (![self connectedToNetwork]) {
        // Show internet is not available message

        [[[UIAlertView alloc]
          initWithTitle:NSLocalizedString(@"FailedToShare.message", nil)
          message:NSLocalizedString(@"NoInternetConnectionText.message", nil)
          delegate:self
          cancelButtonTitle:@"OK"
          otherButtonTitles:nil] show];
        return;

    }

    if ([SLComposeViewController isAvailableForServiceType:SLServiceTypeFacebook]) {

→       SLComposeViewController *mySLComposer = [SLComposeViewController
→   composeViewControllerForServiceType:SLServiceTypeFacebook];

        [mySLComposer setInitialText:comentario];

        [mySLComposer addImage:imagem];

        [mySLComposer setCompletionHandler:^(SLComposeViewControllerResult result) {

            switch (result) {
                case SLComposeViewControllerResultCancelled:
                    NSLog(@"Post Canceled");
                    break;
                case SLComposeViewControllerResultDone:
                    /*
                        // Show success message
                      [[[UIAlertView alloc]
→                      initWithTitle:NSLocalizedString(@"SharedWithSucess.message",
→   nil) message:NSLocalizedString(@"PostSucessful.message", nil) delegate:self
→   cancelButtonTitle:@"OK" otherButtonTitles:nil] show];
                    */
                    break;

                default:
                    break;
            }

        }];

        [self presentViewController:mySLComposer animated:YES completion:nil];
    }
    else {
        [[[UIAlertView alloc]
          initWithTitle:NSLocalizedString(@"NoFacebook.message", nil)
          message:NSLocalizedString(@"NoFacebookText.message", nil)
          delegate:self
          cancelButtonTitle:@"OK"
          otherButtonTitles:nil] show];
    }
}
```

Algoritmo 5: Método para publicar no *Facebook*

5 Resultados

Este capítulo tem como propósito tratar os resultados alcançados com a conclusão deste trabalho, mostrando o alinhamento da proposta e heurísticas de usabilidade com o aplicativo finalizado.

5.1 Alinhamento com a proposta

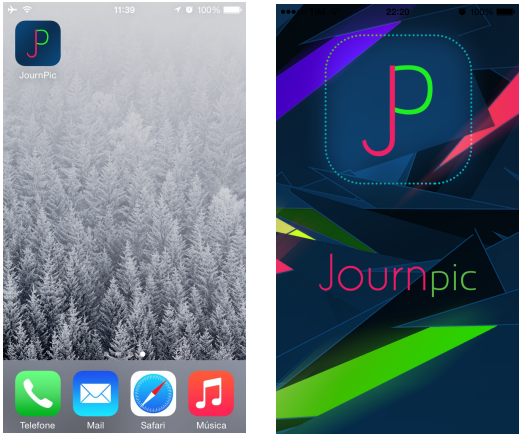
Durante a proposta deste trabalho de conclusão de curso (Capítulo 3) foram indicados os principais aspectos abordados pela aplicação. Um resumo dos mesmos estão listados a seguir.

1. Filtros por palavra-chave, cor e emoção.
2. **Adaptabilidade para vários tamanhos de tela.**
3. Tamanho mínimo de área de toque.
4. Customização de cores e emoções.
5. **Informar a posição no fluxo de telas.**
6. **Internacionalização para inglês e português.**
7. **Compartilhamento no *Facebook* e *Twitter*.**

Os aspectos em negrito já foram tratados anteriormente. Os aspectos "Adaptabilidade para vários tamanhos de tela", "Informar a posição no fluxo de telas" "Internacionalização para inglês e português" e "Compartilhamento no *Facebook* e *Twitter*" foram tratados na Seção 4.3.2, Seção 4.3.3, Seção 4.3.5 e Seção 4.3.6, respectivamente.

O desenvolvimento do Journpic foi concluído atendendo todos esses aspectos. Uma descrição das telas finais do aplicativo e como os aspectos remanescentes foram abordados está presente a seguir.

A Figura 11 apresenta o ícone do aplicativo e a tela de inicialização do mesmo. As cores foram escolhidas de maneira que o contraste entre o ícone do aplicativo e o plano de fundo fosse mantido na maior parte dos casos. Por isso o plano de fundo do ícone é escuro e a cor da fonte é clara.



(a) Ícone do Journpic

(b) Tela de inicialização

Figura 11: Arte do Journpic

A tela principal do aplicativo é apresentada na Figura 12. Esta tela está imbuída em uma *Navigation Controller*, perceptível através da barra superior. Esta barra possui o título "Últimos 30 Dias", isto quer dizer que o filtro aplicado sobre as fotos foi exatamente este. Portanto, a *Collection View*, posicionada logo abaixo dessa barra, apresenta somente as fotos remanescentes após a aplicação do filtro de busca.

O título da *Navigation Controller* se adéqua para exibir o filtro que está sendo aplicado no momento. Por exemplo, se o usuário selecionar o filtro de emoção "Amor", título se tornará este e as fotos se adequarão para exibir somente as que possuem esta emoção.

Figura 12: Tela principal do aplicativo apresentando todas as fotos com o filtro escolhido

Ao tocar em uma das células da *Collection View* presente na tela principal do aplicativo, o usuário é levado a uma nova tela onde é possível visualizar livremente a foto presente na célula selecionada anteriormente (Figura 13). As funcionalidades atreladas aos gestos foram implementadas segundo o padrão do iOS, como listadas a seguir.

- Toque simples - Faz aparecer ou desaparecer a barra de navegação e a barra de ferramentas presentes na parte superior e inferior, respectivamente.
- Toque duplo - Aplica zoom na foto, centralizando a imagem área tocada.
- Deslize - Move a foto no eixo x e y seguindo o deslocamento do dedo.
- Pinça - Aplica ou remove o zoom da foto, partindo do ponto médio entre os dois dedos.

Figura 13: Tela de visualização da fotos escolhida. É possível ampliar a foto por gestos.

Na tela de visualização de foto exite um botão chamado “Detalhes”. Ao clicar nele, a imagem da tela é desfocada e aparecem nítidas são: a emoção, a tag e a descrição (Figura 14). Ista tela foi implementada desta maneira para seguir uma das heurísticas de usabilidade, a qual afirma que as informações mais importantes devem ter enfoque na tela e que informações adicionais podem competir pela atenção do usuário. Os menores botões presentes nesta tela se encontram na barra inferior e possuem um tamanho de exatos 44 pontos (44 pontos é o tamanho mínimo aceitável para botões em aplicações iOS (APPLE, 2014b)), atendendo o aspecto remanescente “Tamanho mínimo de área de toque”.

Nesta tela não há uma saída explícita, mas qualquer toque faz com que o usuário retorne à tela anterior.

A Figura 15 apresenta a tela de adição e edição de uma foto. Esta tela pode ser acessada após tirar ou selecionar uma nova foto do rolo da câmera ou clicando no botão de editar presente na tela de visualização da foto (Figura 14). Cooper, Reimann e Cronin (2007) explicam que, em dispositivos móveis, devesse restringir o uso de teclado o máximo possível. Dado que cada emoção tem uma cor atrelada a ela, foram utilizadas bolas coloridas com a cor da respectiva emoção imbuídas em uma *Collection View* com fluxo lateral. O canal de cor alfa de cada uma das bolas foi levemente alterado para ficar translúcido, e ao tocar sobre quaisquer uma delas, a borda da bola (emoção) seleciona

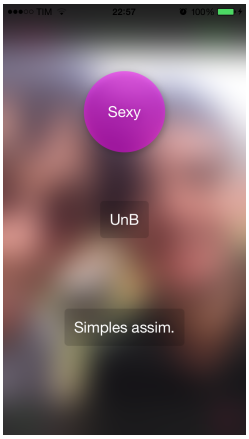


Figura 14: Tela com os detalhes da foto, apresentando a emoção, a tag e a descrição.

retornar ao alfa completamente opaco. Isto gera contraste e informa ao usuário qual emoção está selecionada. Os demais campos de texto desta tela são autoexplicativos.

Figura 15: Tela com os campos de emoção, tag e descrição da foto.

Na tela principal do aplicativo (Figura 12) existe um botão posicionado no canto superior esquerdo, na mesma posição que estão presentes os botões de "Voltar" em todas as telas mais abaixo no fluxo hierárquico. Este botão está posicionado neste local porque revela o menu de nível mais alta na hierarquia do aplicativo, apresentando opções de configuração e filtros (Figura 16).

O guia de interface apresentado pela Apple (2014b) não aconselha o *Branding* excessivo. Isto quer dizer que o usuário provavelmente não se esqueceu do aplicativo que está usando, por isso não é adequado exibir o logotipo da empresa ou do aplicativo em várias telas e consumir parte valiosa dela. Contudo, é aconselhado o uso das cores do aplicativo para tematização do mesmo, usando suas cores nos elementos de interface gráfica. O Journpic utiliza a cor vermelha do ícone em botões de ação destrutiva, como retorno ou apagar, e a cor verde do ícone em botões de ação construtiva, como tirar foto ou compartilhar.

O menu principal do aplicativo, retratado na Figura 16, apresenta uma pequena logomarca que não consome lugar de outros botões, e portanto não impacta na utilização da interface. Nesta tela também fica evidente os filtros de busca por tag e emoção (as cores estão atreladas às emoções), atendendo o aspecto "Filtros por palavra-chave, cor e emoção".

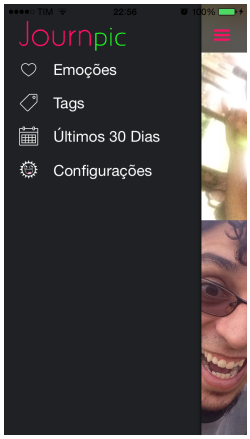


Figura 16: Tela cuja implementação foi feita para que parecesse estar por trás da tela principal, revelando um menu

A Figura 17 apresenta todas as emoções registradas. Esta tela pode ser acessada ao clicar na célula de Emoções presente no menu principal do aplicativo.

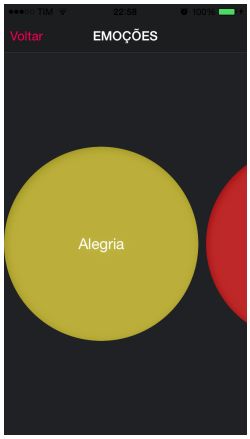


Figura 17: Tela com todos os filtros de emoções

Os emoções apresentadas neste filtro são compostas das emoções padrões do aplicativo (alegria, amor, raiva, tranquilidade e tristeza) e das emoções criadas pelo usuário.

Semelhante à tela de filtro por emoções, a tela de filtro por tags pode ser acessada através do menu principal. Esta tela está retratada na Figura 18.

A tela de configurações (Figura 19) permite o usuário acessar duas áreas passíveis de configuração. A primeira leva à tela de customização de emoções, retratada pela Figura 20, e a segunda leva às configurações de marca d'água no compartilhamento de uma foto, apresentado na Figura 22.

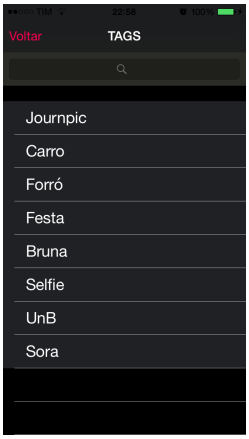


Figura 18: Tela para selecionar o filtro das fatos dadas as suas tags

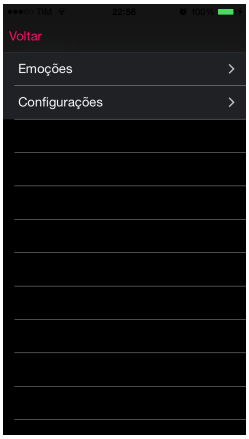


Figura 19: Tela final com os ítens de configuração

A Figura 20 mostra todas as emoções padrões do aplicativo e todas as emoções customizadas. O usuário pode tocar qualquer uma das emoções para alterar seus parâmetros ou tocar o botão de "Adicionar Emoção Customizada" para criar uma nova emoção com o nome e a cor desejada.

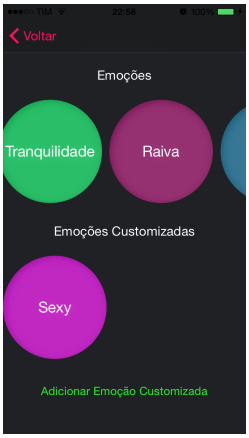


Figura 20: Tela com todas as emoções padrões e emoções criadas pelo usuário

A Figura 21 apresenta ao usuário os dois atributos de uma emoção: nome e cor. O usuário pode alterar estes atributos para que retratem melhor seu gosto pessoal. A customização destes atributos contempla o último aspecto remanescente - “Customização de cores e emoções”.

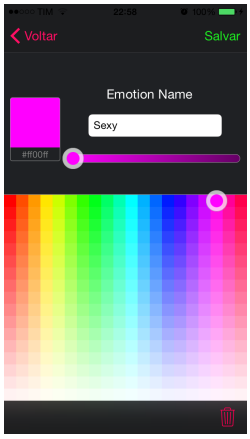


Figura 21: Tela com os ítens customizáveis de uma emoção

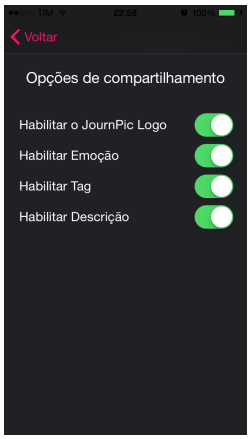
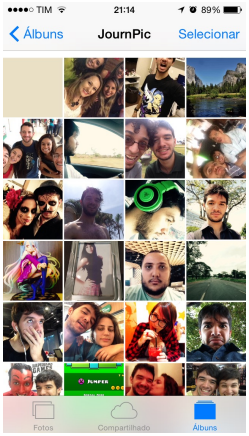
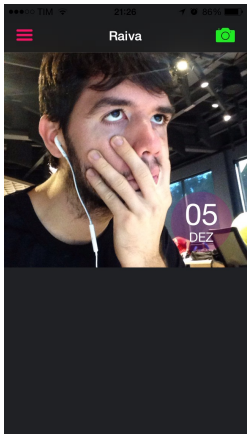


Figura 22: Tela com as configurações de marca d’água para publicar uma foto.

Treisman e Gelade (1980) diz que quanto maior o número de itens a serem focados visualmente, maior será o tempo para percorrer todos eles. Ao aplicar filtros de busca com o uso do aplicativo, o número de itens a serem focados diminui, e consequentemente o tempo. Por exemplo, se um usuário deseja encontrar uma foto que transmita a emoção “Raiva” utilizando o aplicativo nativo de fotos do sistema iOS, ele terá de observar as fotos uma a uma até que encontre a foto desejada. Ao aplicar um filtro neste mesmo álbum utilizando o aplicativo desenvolvido, o número de fotos é reduzido, optimizando o tempo total necessário para que tomaria para olhar todas uma a uma. Esta situação está retratada na Figura 23.



(a) Álbum do aplicativo de fotos do iOS



(b) Aplicação de filtro por fotos com a emoção Raiva

Figura 23: Comparação entre o número de fotos exibidas no aplicativo nativo de fotos do sistema iOS o aplicativo Journpic

5.2 Heurísticas de usabilidade

Esta seção demonstra como as heurísticas de usabilidade listadas no Capítulo 2 foram aplicadas neste projeto. Visando uma melhor visualização de todos as medidas de usabilidade tomadas durante o desenvolvimento do projeto, a seguir está a lista contendo cada heurística e como cada uma foi tratada.

1. Visibilidade do estado do sistema - Existem dois fatores que podem interromper o funcionamento esperado do aplicativo. O primeiro destes fatores é o nível de bateria, pelo simples fato de que um dispositivo móvel poder ter sua bateria totalmente drenada e cessar seu funcionamento. O segundo fator é a conectividade com a Internet, que pode interromper a funcionalidade de compartilhamento de fotos. Por isso o aplicativo deixa evidente o nível de bateria em todas as telas e o sinal de rede na parte superior da tela, informações providas pelo sistema operacional.

2. Compatibilidade entre o sistema e o mundo real -Atributos de permissão de acesso, identificadores de tipos de mídia e referências aos itens da biblioteca do dispositivo não são comuns ao usuário, e portanto a terminologia utilizada com o usuário neste aplicativo foi simplesmente “foto”. Desta maneira o usuário é poupado dos termos técnicos da programação e lida com um termo mais cotidiano, do mundo real. Para tornar mais claro o entendimento do aplicativo, foi feita também a internacionalização do aplicativo em 2 línguas: inglês e português (explicado na Seção 4.3.5).

3. Controle do usuário e liberdade - O aplicativo foi desenvolvido sob uma estrutura de *Navigation Controller*. O fluxo de telas do aplicativo segue um fluxo de hierarquia da esquerda para a direita, sendo a esquerda o nível mais alto da hierarquia e a

direita o nível mais baixo. Portanto, em qualquer tela do aplicativo, será possível voltar, usar uma ”saída de emergência“, sempre presente no canto superior esquerdo do aplicativo.

4. Consistência e padrões - A linguagem adotada pelo aplicativo segue a terminologia do próprio sistema. Mesmo que a proposta do aplicativo se remeta a memórias, lembranças, lugares, a linguagem utilizada sempre utiliza o termo “foto”. Foi tomado o cuidado de utilizar as mesmas palavras, e não sinônimos, ao longo do aplicativo. Paralelamente, as ações padrões de gestos, o tamanho e estilo das fontes, assim como demais aspectos de interface foram implementados de acordo com os padrões iOS descrito nos guia Apple (2014b).

5. Prevenção de erros - Existem mensagens de confirmação sempre que o usuário está prestes a executar uma ação permanente, como apagar uma foto, prevenindo possíveis erros. Fora isso, a configuração de conta das redes sociais pode ser feita nos próprios ajustes do sistema operacional, evitando que o usuário necessite fazer login durante a utilização do aplicativo.

6. Reconhecimento ao invés de recordação - Ao navegar pelo fluxo de telas do aplicativo, somente as opções e dados importantes são mostrados ao usuário. Por exemplo, ao selecionar uma foto, o usuário é levado a uma tela onde a foto é ampliada e toma a maior parte da tela, sendo a informação em evidência, e opções como editar e compartilhar ocupam porções muito menores desta mesma tela. Processos semelhantes ocorrem quando o usuário está selecionando um filtro por emoção ou por tag.

7. Flexibilidade e eficiência de uso - A ausência de um teclado físico em dispositivos iOS dificulta a implementação de atalhos tradicionais em outros sistemas. Gestos como a mão, como pinça, deslizar dedos, toque longo, poderiam ser utilizados para acessar, por exemplo, as configurações do aplicativo, mas não foram implementadas porque não são normalmente usados com esta finalidade e quebrariam o fluxo hierárquico do aplicativo.

8. Estética e design minimalista - As informações importantes em cada tela sempre ocupam a maior porção da tela. Ações e opções adicionais, como exclusão e edição são representadas por botões menores, seguindo o princípio de que informações mais relevantes devem estar mais visíveis. Jones e Marsden (2006) explicam que uma maneira comum de aprimorar menus é utilizando ícones, já que são utilizados para substituir, ou destacar, descrições textuais e funcionalidades. Neste aplicativo foram usados ícones tanto para substituir textos descritivos de ações comuns, como o ícone de câmera para tirar fotos e o ícone de lixeira para apagar, quanto para destacar textos, como o ícone de coração do lado do texto de emoções e o ícone de engrenagem do lado de texto de configurações.

9. Ajude usuários a reconhecer, diagnosticar e se recuperar de erros - O aplicativo demonstra as possíveis causas de um problema ou ações a serem tomadas sempre que uma mensagem de erro aparece na tela. As mensagens consistem basicamente em conectividade, nível de bateria e inserção de fotos duplicadas. Por exemplo, causo um usuário tente publicar uma foto no *Facebook* mas a ação não é concluída, o aplicativo mostra uma mensagem de erro avisando que a foto não pode ser publicada e convida o usuário a checar se a conexão com a Internet está funcionando corretamente.

10. Ajuda e documentação - O aplicativo não possui manual, mas ao publicar o aplicativo na Apple Store, foi fornecido um site para suporte técnico. Desta maneira um usuário consegue enviar possíveis dúvidas ao desenvolvedor.

6 Conclusão

Este trabalho de conclusão de curso apresenta a criação de um aplicativo de fotos para iOS com funcionalidades de busca.

Foram aplicadas técnicas de Engenharia de Software para o desenvolvimento do projeto (as áreas são descritas no início do Capítulo 4). As técnicas utilizadas auxiliaram e guiaram o desenvolvimento, possibilitando que o projeto fosse concluído com êxito.

Foram apontados os principais conceitos de usabilidade utilizados no projeto. A descrição de como cada heurística de usabilidade proposta por Nielsen (1994) foi aplicada no desenvolvimento do aplicativo pode ser vista em detalhes nos resultados (Capítulo 5).

Existiu a necessidade de que a interface gráfica do aplicativo desenvolvido entre em concordância com os padrões definidos no guia de interface humana Apple (2014b), caso contrário o a aplicativo não poderia ser publicado na AppStore. O desenvolvimento do aplicativo seguiu à risca as orientações deste guia, e a prova disto é que o aplicativo foi publicado com sucesso nessa loja virtual. É importante ressaltar que o aplicativo está disponível de graça.

A persistência de dados permite ao usuário gravar suas fotos em um álbum separado no sistema. Esta funcionalidade possibilita que o usuário tenha acesso às fotos mesmo que o aplicativo tenha sido excluído ou atualizado. Por se tratarem de fotos, que são arquivos pessoais, é importante que a persistência seja mantida para que o usuário não perca dados acidentalmente.

Como principal restrição do trabalho tem-se o fato de não haver acesso suficiente ao sistema iOS. Por motivos de segurança, os aplicativos podem ler alguns dados externos do próprio sistema, como álbuns, login do *Facebook* e preferências de acessibilidade, mas não têm permissão para alterá-los. Por isso, não é possível excluir fotos do álbum, somente sua referência no aplicativo. Mesmo que um usuário exclua uma foto utilizando o aplicativo Journpic, ela deverá ser removida manualmente através do aplicativo de fotos do sistema para que seja apagada completamente.

6.1 Trabalhos futuros

Os fatores de usabilidade foram tratados baseados em seus conceitos, mas não foram feitos estudos para avaliá-los. Sugere-se como trabalho futuro que seja feito uma pesquisa de campo verificando estes aspectos.

Existem vários melhorias de que podem ser aplicadas nas funcionalidades já existentes no Journpic, desde a possibilidade de publicação em outras redes sociais, melhores algoritmos para tratar gestos ao manusear fotos, até a refatoração do banco de dados do aplicativo para que funcione com o *Core Data* ao invés do SQLite.

Para este projeto sugere-se a implementação de novas funcionalidades, pontos de extensões, que podem servir como experimentos sociais. Uma delas seria adicionar a possibilidade de publicar ou reproduzir todas as imagens de uma determinada emoção uma a uma, pois fotos aparentemente tranquilas ou felizes podem não ter o mesmo efeito em todas as pessoas. Por exemplo, uma foto tirada enquanto pratica queda livre pode ter a emoção de "Alegria" mas pode simbolizar puro pavor para outras.

Outro ponto para extensão é adicionar o suporte a vídeos no aplicativo como base para uma nova entrada no diário, assim como é o papel das fotos no estado atual do aplicativo.

Finalmente, uma ótima oportunidade de extensão seria possibilitar que o aplicativo pudesse se integrar com repositórios externos como o *iCloud*, *Google Drive* e o *Dropbox*. Desta maneira o usuário conseguiria gravar grupos inteiros de entradas no diário para serem acessadas em outros dispositivos.

Referências

APPLE INC. *App Programming Guide for iOS*. Cupertino, EUA, 2014. 118 p. Citado 8 vezes nas páginas 46, 87, 88, 89, 90, 91, 93 e 94.

APPLE INC. *iOS Human Interface Guidelines*. Cupertino, EUA, 2014. 228 p. Citado 6 vezes nas páginas 34, 50, 59, 60, 65 e 67.

BASILI, V.; BRIAND, L.; MELO, W. A validation of object-oriented design metrics as quality indicators. *Software Engineering, IEEE Transactions on*, 1996. v. 22, n. 10, p. 751–761, Oct 1996. ISSN 0098-5589. Citado na página 43.

BERTOLINO, A. Software testing research: Achievements, challenges, dreams. In: *2007 Future of Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007. (FOSE '07), p. 85–103. ISBN 0-7695-2829-5. Disponível em: <http://dx.doi.org/10.1109-/FOSE.2007.25>. Citado na página 40.

BUSCHMANN, F. et al. *Pattern-Oriented Software Architecture, A System of Patterns*. Wiley, 2013. (Pattern-Oriented Software Architecture). ISBN 9781118725269. Disponível em: <https://books.google.com.br/books?id=j_ahu_BS3hAC>. Citado na página 45.

COOPER, A.; REIMANN, R.; CRONIN, D. *About Face 3: The Essentials of Interaction Design*. Wiley, 2007. ISBN 9780470084113. Disponível em: <http://books.google.com-.br/books?id=9F7gaZKd2rYC>. Citado na página 59.

EMARKETER. Mobile internet access in brazil to drive social usage. 2014. 2014. Disponível em: <http://www.emarketer.com/Article/Mobile-Internet-Access-Brazil-Drive-Social-Usage/1011198>. Acesso em: 10.11.2014. Citado na página 34.

FONSECA, I. *Engenharia de Software Conference*. São Paulo, SP, Brasil: [s.n.], 2009. Citado 2 vezes nas páginas 39 e 40.

GALITZ, W. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. Wiley, 2007. (Wiley Desktop Editions Series). ISBN 9780470146224. Disponível em: <http://books.google.com.br-/books?id=Q3Xp_Awu49sC>. Citado na página 24.

GAZZALEY, A. *How Mobile Tech Can Influence Our Brain*. 2012. Disponível em: <http://edition.cnn.com/2012/09/23/opinion/gazzaley-mobile-brain/>. Acesso em: 18.10.2014. Citado 2 vezes nas páginas 24 e 34.

JONES, M.; MARSDEN, G. *Mobile Interaction Design*. Wiley, 2006. ISBN 9780470090893. Disponível em: <https://books.google.com.br-/books?id=pQxTAAAAMAAJ>. Citado na página 65.

KRUCHTEN, P. *The Rational Unified Process: An Introduction*. Addison-Wesley, 2004. (The Addison-Wesley object technology series). ISBN 9780321197702. Disponível em: <http://books.google.com.br/books?id=RYCMx6o47pMC>. Citado na página 40.

MONTOLIU, R.; GATICA-PEREZ, D. Discovering human places of interest from multimodal mobile phone data. In: *Proceedings of the 9th International Conference on Mobile and Ubiquitous Multimedia*. New York, NY, USA: ACM, 2010. (MUM '10), p. 12:1–12:10. ISBN 978-1-4503-0424-5. Disponível em: <http://doi.acm.org/10.1145-/1899475.1899487>. Citado na página 23.

NIELSEN, J. *Usability Engineering*. Morgan Kaufmann, 1994. (Interactive Technologies Series). ISBN 9780125184069. Disponível em: <http://books.google.com.br-/books?id=95As2OF67f0C>. Citado 4 vezes nas páginas 24, 29, 30 e 67.

NIELSEN, J. Usability 101: Introduction to usability. 2012. 2012. Disponível em: <http://www.nngroup.com/articles/usability-101-introduction-to-usability>. Acesso em: 16.10.2014. Citado 4 vezes nas páginas 29, 30, 31 e 32.

NUNES, I. D.; CORREIA, R. S. A importância da usabilidade no desenvolvimento de aplicativos para dispositivos móveis. 2013. 5º Encontro de Produção Acadêmico-Científico, 2013. Disponível em: <http://www.cesed.br/enpac/anais/arquivos/anais/areatematica-sistemas/sis001.pdf>. Citado na página 25.

OINAS-KUKKONEN, H. Developing successful mobile applications. In: *Computer Science and Technology*. [S.l.: s.n.], 2003. Citado na página 23.

PRESSMAN, R. *Engenharia de Software*. McGraw Hill Brasil, 2011. ISBN 9788580550443. Disponível em: <https://books.google.com.br/books?id=y0rH9wuXe68C>. Citado na página 45.

SHAW, M.; GARLAN, D. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996. (An Alan R. Apt book). ISBN 9780131829572. Disponível em: <https://books.google.com.br/books?id=5KlQAAAAMAAJ>. Citado na página 45.

SOFTWARE ENGINEERING: GUIDE TO SOFTWARE ENGINEERING BODY OF KNOWLEDGE. *Standards South Africa*. [S.l.], 2007. Disponível em: <http:/-/books.google.com.br/books?id=OAOkGAAACAAJ>. Citado 2 vezes nas páginas 39 e 45.

TORRES, N. *Competitividade empresarial com a tecnologia de informação*. Makron Books, 1995. ISBN 9788534604123. Disponível em: <http://books.google.com.br-/books?id=QBCiAQAACAAJ>. Citado na página 23.

TREISMAN, A. M.; GELADE, G. A feature-integration theory of attention. *Cognitive Psychology*, 1980. v. 12, p. 97–136, 1980. Citado na página 63.

Apêndices

APÊNDICE A – Modelos de Casos de Uso

A seguir é apresentado um diagrama de casos de uso do aplicativo. Este diagrama tem a finalidade de mostrar as principais ações que o usuário pode executar no aplicativo.

A.1 Visão geral dos casos de uso e atores

A Figura 24 mostra que o usuário do aplicativo pode manter (incluir, pesquisar, alterar e excluir) entradas no aplicativo. Para incluir uma entrada no diário é necessário tirar ou escolher uma foto. O usuário também tem a opções de compartilhar as entradas do diário nas redes sociais *Facebook* e *Twitter*. Assim como as entradas do diário, o usuário também pode manter emoções.

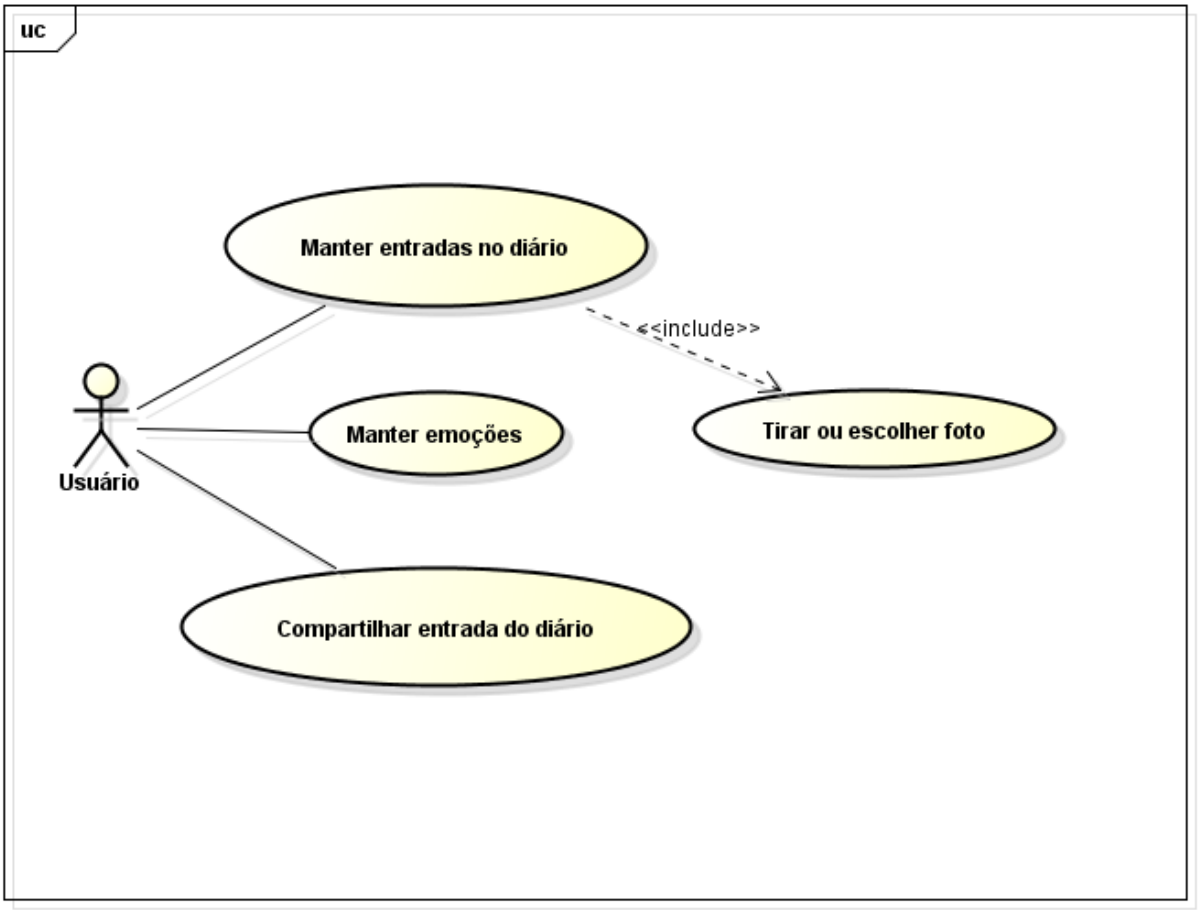


Figura 24: Visão geral do dos casos de uso e atores

A.2 Descrição dos casos de uso

Uma descrição detalhada de cada passo presente no fluxo de cada um dos casos de uso apresentados pode ser vista a seguir.

A.2.1 UC - 01 Manter entradas no diário

1. Manter entradas no diário - Breve descrição

O aplicativo Journpic permitirá a inclusão, alteração, pesquisa e exclusão de entradas no diário (Uma entrada no diário é composta por foto, emoção e tag).

2. Atores

Usuário

3. Pré-condições

3.1 Não há pré-condições.

4. Fluxo de eventos

4.1 Fluxo básico - Manter entradas no diário

(P1) O usuário toca o botão de tirar foto na tela principal.

(P2) O sistema exibe as opções de tirar uma foto ou selecionar uma já existente.

(P3) O usuário seleciona ou tira uma foto. [E01]

(P4) O sistema exibe a tela com os campos dos dados do diário.

(P5) O usuário preenche os campos.

(P6) O sistema valida os campos preenchidos.

(P7) O sistema habilita o botão de salvar.

(P8) O usuário salva a nova entrada no diário.

(P9) O sistema insere os dados no banco de dados.

(P10) Fim do caso de uso.

4.2 Fluxo Alternativo - Alterar entrada no diário

(A01.1) O usuário realiza o fluxo alternativo [A03]

(A01.2) O usuário toca na entrada do diário desejada.

(A01.3) O sistema mostra a tela com a foto da entrada selecionada.

(A01.4) O usuário toca o botão de editar.

(A01.5) O sistema carrega os dados a partir do banco de dados.

(A01.6) O sistema exibe a tela com os campos dos dados do diário.

(A01.7) O usuário altera os campos desejados.

(A01.8) O sistema valida os campos.

(A01.9) O usuário salva a entrada do diário.

(A01.10) O sistema altera os dados no banco de dados.

(A01.11) O caso de uso é encerrado.

4.3 Fluxo Alternativo - Deletar entrada do diário

(A02.1) O usuário realiza o fluxo alternativo [A03]

(A02.2) O usuário toca na entrada do diário desejada.

(A02.3) O sistema mostra a tela com a foto da entrada selecionada.

(A02.4) O usuário toca o botão de excluir.

(A02.5) O sistema solicita a confirmação da ação.

(A02.6) O usuário confirma a ação.

(A02.7) O sistema remove a referência da foto.

(A02.8) O caso de uso é encerrado.

4.4 Fluxo Alternativo - Pesquisar entrada no diário

(A03.1) No passo (P1) do fluxo básico, o usuário seleciona uma das opções de filtros de busca: Emoções, Tags ou Últimos 30 dias.

(A03.1.1.1) O usuário selecionou o filtro por emoções.

(A03.1.1.2) O sistema carrega do banco as emoções cadastradas.

(A03.1.1.3) O sistema exibe uma tela com as emoções.

(A03.1.1.4) O usuário seleciona uma das emoções.

(A03.1.1.5) O sistema faz carrega as fotos no banco utilizando este filtro.

(A03.1.2.1) O usuário selecionou o filtro por tags.

(A03.1.2.2) O sistema carrega do banco as tags cadastradas.

(A03.1.2.3) O sistema exibe uma tela com as tags.

(A03.1.2.4) O usuário seleciona uma das tags.

(A03.1.2.5) O sistema faz carrega as fotos no banco utilizando este filtro.

(A03.2) O sistema exibe o resultado da pesquisa usando o filtro selecionado.

(A03.3) O caso de uso é encerrado.

4.5 Fluxo Exceção - E01

No (P3) o usuário selecionou uma foto já existente no álbum Journpic.

E01.1 O sistema apresenta a mensagem “Não é possível adicionar a foto, pois ela já está no álbum Journpic”

E01.2 O sistema retorna ao passo (P1).

5. Pós-condições

5.1 Após a execução deste caso de uso, a entrada do diário deverá: ter sido inserida com sucesso/ter seus dados alterados com sucesso/ter sido removida com sucesso.

5.2 O sistema exibirá a tela inicial do aplicativo.

5.3 Os dados estarão atualizado no banco de dados.

A.2.2 UC - 02 Manter emoções

1. Manter emoções - Breve descrição

O aplicativo Journpic permitirá a inclusão, alteração e exclusão de emoções (Uma emoção é composta por nome e cor).

2. Atores

Usuário

3. Pré-condições

3.1 Não há pré-condições.

4. Fluxo de eventos

4.1 Fluxo básico - Inserir uma emoção

(P1) O usuário toca o botão de configurações na tela principal.

(P2) O sistema exibe seus itens configuráveis.

(P3) O usuário seleciona a opção de emoções.

(P4) O sistema carrega as emoções a partir do banco de dados.

(P5) O sistema exibe a tela com as emoções.

(P6) O usuário toca o botão de "Adicionar Emoção Customizada".

(P7) O sistema exibe a tela com os elementos para criação de uma nova emoção.

(P8) O usuário preenche os campos.

(P9) O sistema valida os campos preenchidos. [E01]

(P10) O usuário salva a nova emoção.

(P11) O sistema insere os dados no banco de dados.

(P12) Fim do caso de uso.

4.2 Fluxo Alternativo - Alterar emoção

(A01.1) No passo (P6) do fluxo básico, o usuário toca na emoção que deseja alterar.

(A01.2) O sistema exibe a tela com os elementos para edição da emoção.

(A01.3) O usuário altera os campos.

(A01.4) O sistema valida os campos preenchidos. [E01]

(A01.5) O usuário salva as alterações.

(A01.6) O sistema altera os dados no banco de dados.

(A01.7) O caso de uso é encerrado.

4.3 Fluxo Alternativo - Deletar emoção

(A02.1) No passo (P6) do fluxo básico, o usuário toca na emoção customizada que deseja deletar.

(A02.2) O sistema exibe a tela com os elementos para edição da emoção.

(A02.3) O usuário toca a botão de excluir.

(A02.4) O sistema remove a emoção no banco de dados.

(A02.5) O caso de uso é encerrado.

4.4 Fluxo Exceção - E01

Em (P9)(A01.4) o usuário tentou salvar/alterar uma emoção com o nome contendo menos de 2 caracteres.

E01.1 O sistema apresenta a mensagem "O campo de emoção deve ter ao menos 2 caracteres"

E01.2 O sistema retorna ao passo anterior.

5. Pós-condições

5.1 Após a execução deste caso de uso, a emoção deverá: ter sido criada com sucesso/ter seus dados alterados com sucesso/ter sido removida com sucesso.

5.2 O sistema exibirá a tela com as emoções cadastradas.

5.3 Os dados estarão atualizado no banco de dados.

A.2.3 UC - 03 Compartilhar entrada do diário

1. Compartilhar entrada do diário - Breve descrição

O aplicativo Journpic permitirá o compartilhamento de entradas do diário nas redes sociais Facebook e Twitter.

2. Atores

Usuário

3. Pré-condições

3.1 Uma entrada do diário foi selecionada.

4. Fluxo de eventos

4.1 Fluxo básico - Compartilhar

(P1) O usuário toca o botão de compartilhar.

(P2) O sistema exibe as opções de compartilhar no facebook ou twitter.

(P3) O usuário seleciona a opção desejada.

(P4) O sistema exibe a tela com opções de compartilhamento da rede social. [E01]

(P5) O usuário seleciona as opções de compartilhamento.

(P6) O usuário toca o botão de Publicar. [E01]

(P7) O sistema exibe um som de confirmação.

(P8) Fim do caso de uso.

4.2 Fluxo Exceção - E01

Em (P4)(P6) o sistema detecta que não há conexão com a Internet.

E01.1 O sistema apresenta a mensagem "Você não pode fazer uma postagem agora. Verifique sua conexão com a Internet."

E01.2 O sistema retorna ao passo (P1).

5. Pós-condições

5.1 Após a execução deste caso de uso, a entrada do diário deverá ter sido compartilhada com sucesso.

5.2 O sistema exibirá a tela com as foto da entrada do diário.

APÊNDICE B – Arquivos XIB do Journpic

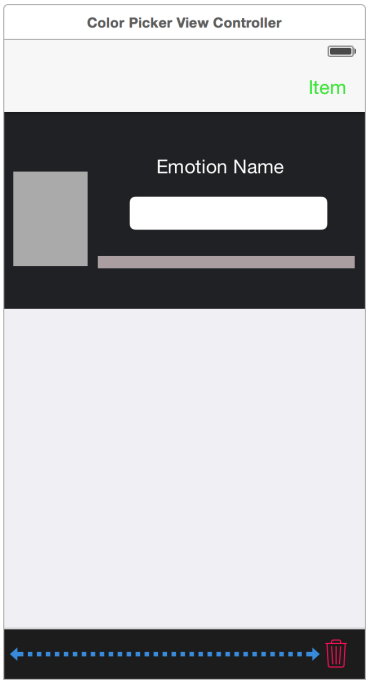


Figura 25: JPColorPickerViewController.xib

Figura 26: JPImageDetailsViewController.xib

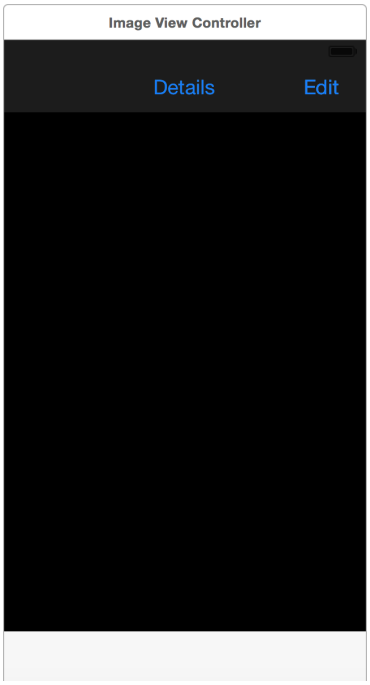


Figura 27: JPImageViewController.xib

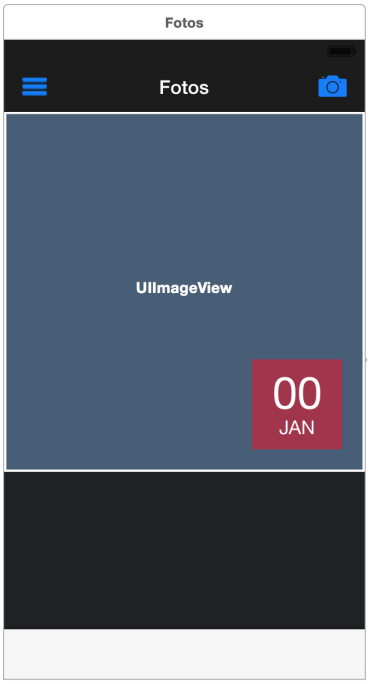


Figura 28: JPMainCollectionViewController.xib

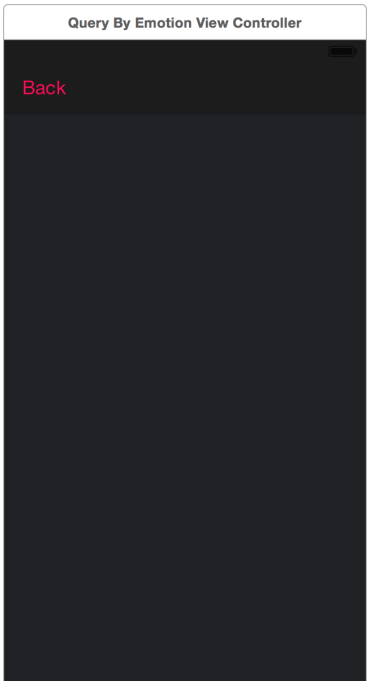


Figura 29: JPQueryByEmotionViewController.xib

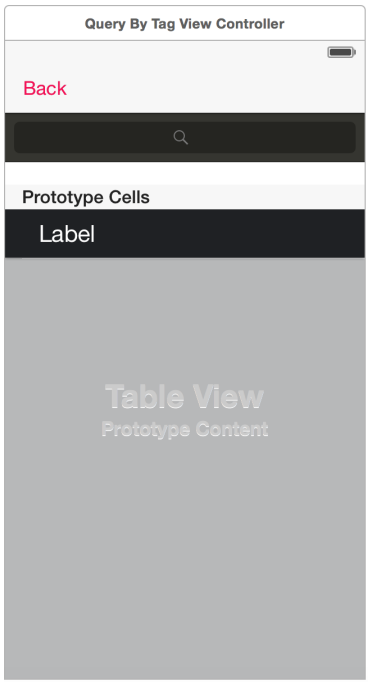


Figura 30: JPQueryByTagViewController.xib

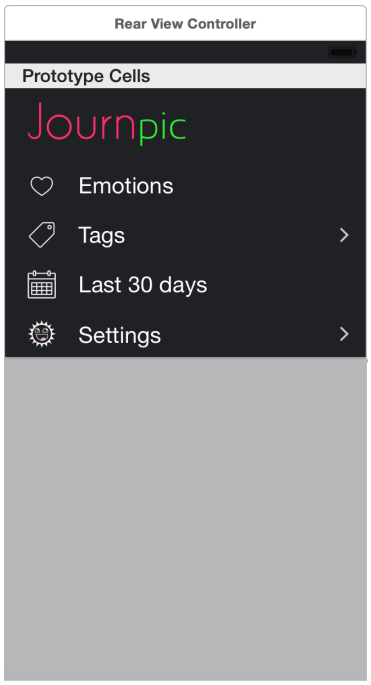


Figura 31: JPRearViewController.xib

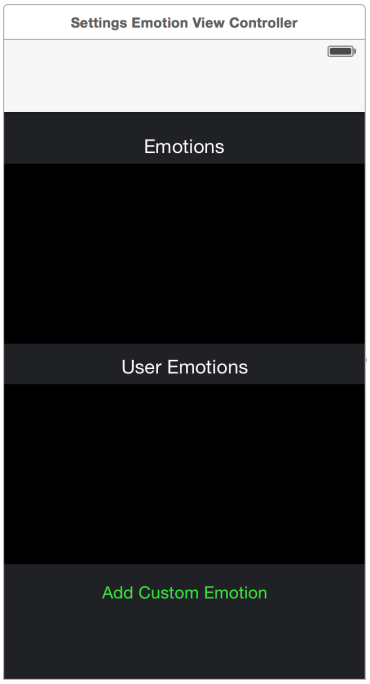


Figura 32: JPSettingsEmotionViewController.xib

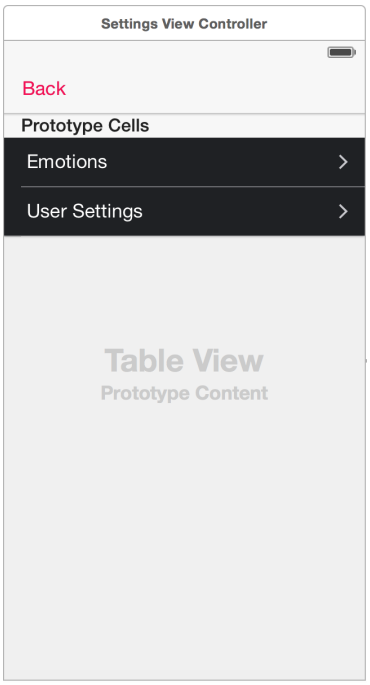


Figura 33: JPSettingsViewController.xib

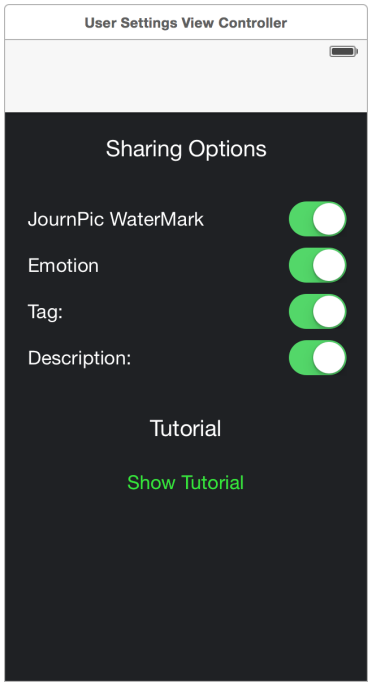


Figura 34: JPUserSettingsViewController

Anexos

ANEXO A – Ciclo de Vida de um Aplicativo iOS

O guia App Programming Guide for iOS (APPLE, 2014a) define um aplicativo como sendo uma interação sofisticada entre código customizado e os frameworks do sistema, já que os frameworks provêm a infraestrutura básica que todos os aplicativos precisam para rodar e o programador provê o código necessário para customizar essa infraestrutura e dar à aplicação a aparência e o comportamento desejados.

É importante apontar que os frameworks iOS se baseiam em padrões de projeto como MVC e delegação em suas implementações.

A.1 Função main

Como reafirmado em (APPLE, 2014a), o ponto de partida para qualquer aplicação baseada na Linguagem C é a função *main* e para o desenvolvimento de aplicativos iOS o mesmo se aplica, já que a linguagem nativa é o *Objective-C*. O que difere, no entanto, é que em aplicativos o programador não tem que escrever a função. O ambiente de desenvolvimento XCode cria esta função como parte de um projeto básico, como exemplificado no Algoritmo 6.

main.m

```
1 #import <UIKit/UIKit.h>
2 #import "AppDelegate.h"
3
4 int main(int argc, char * argv[])
5 {
6     @autoreleasepool {
  →       return UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate
7 →   class]));
8     }
9 }
```

Algoritmo 6: Função *main* em aplicativos iOS

Portanto, a única finalidade da função *main* é delegar o trabalho para o framework do UIKit. A função *UIApplicationMain* maneja este processo através da criação de core objects do aplicativo, carregando a interface do usuário contida nos arquivos de storyboard, chamando qualquer código customizado feito pelo desenvolvedor a fim de mudar quaisquer valores iniciais, e por fim inicia o laço principal de execução. Logo, as únicas

tarefas que estão por conta do desenvolvedor são fornecer os arquivos de storyboard e qualquer código de inicialização customizada do aplicativo.

A.2 Estrutura de um aplicativo

Segundo o guia (APPLE, 2014a), durante o lançamento, a função *UIApplication-Main* configura e carrega vários objetos e inicia o ciclo principal do aplicativo. No núcleo de cada aplicativo iOS está o objeto *UIApplication*, cuja tarefa é facilitar a interação entre o sistema e outros objetos do aplicativo. A Figura (35) mostra os objetos mais comummente presentes nos aplicativos, enquanto a Tabela (5) lista o papel de cada um desses objetos. Nota-se que os aplicativos iOS usam uma arquitetura MVC. Este padrão de projeto separa os dados do aplicativo e a lógica de negócio da apresentação visual dos dados. Essa arquitetura é crucial para o desenvolvimento de aplicativos que podem rodar em diferentes dispositivos com diferentes tamanhos de tela.

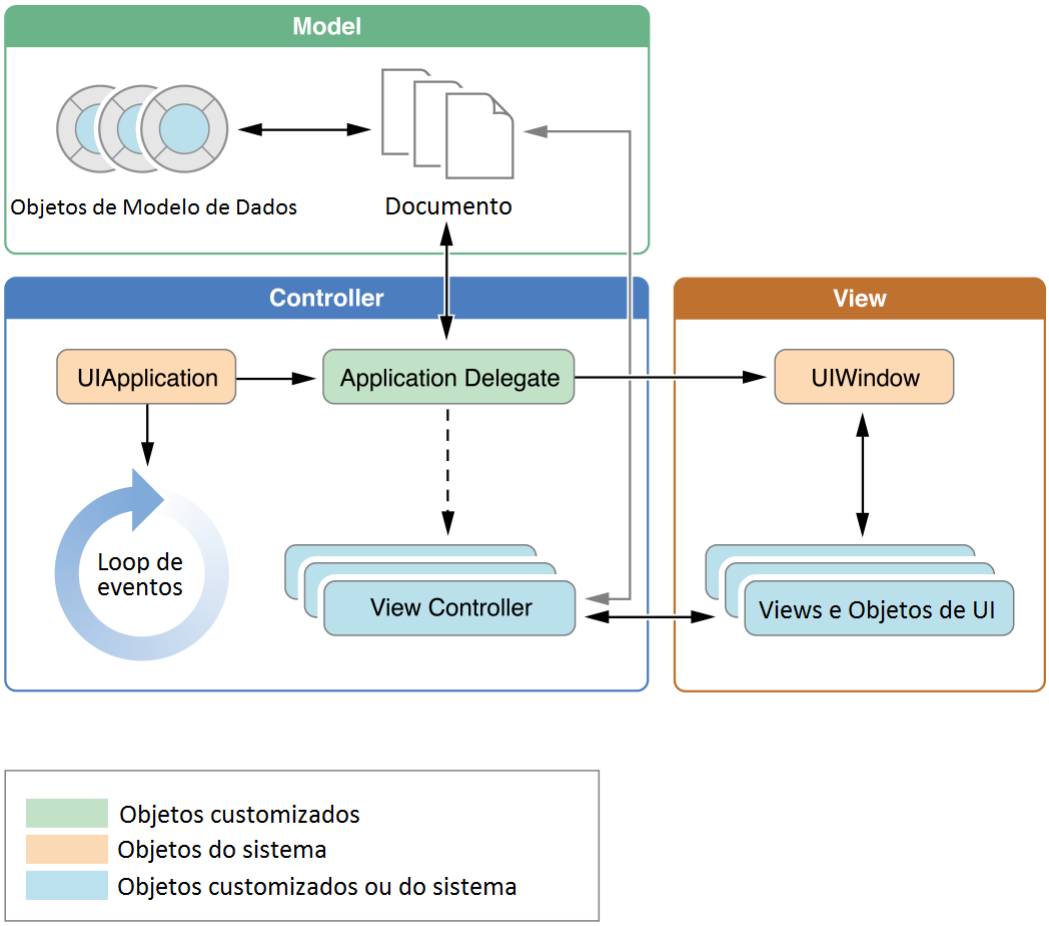


Figura 35: Objetos chave em um aplicativo iOS
Fonte: Adaptado de (APPLE, 2014a)

O que distingue os aplicativos iOS entre si são os dados que eles gerenciam (e a lógica de negócio correspondente) e como ele apresenta estes dados ao usuário. A maior

Objeto	Descrição
UIApplication	Gerencia o loop de eventos e outros comportamentos em alto nível do aplicativo. Também reporta transições chave do aplicativo e eventos especiais (como notificações push) ao seu delegate, que é um objeto customizado.
App Delegate	É o núcleo do código customizado. Este objeto funciona paralelamente com o UIApplication para manejar a inicialização do aplicativo, estados de transição, e outros eventos em alto nível. Este objeto também é o único que deve estar presente tem todos os aplicativos, e por isso é usado para inicializar estruturas de dados.
Modelo de dados	Estes objetos guardam o conteúdo do aplicativo e são específicos de cada aplicativo. Como exemplo, um aplicativo de banco poderia guardar uma base de dados contendo as transações do usuário.
Documentos	Aplicativos também podem utilizar objetos de documento (Subclasse customizada de UIDocument) para manejar parte ou todos seus objetos de modelo de dados. Objetos de documento não são obrigatórios, mas oferecem uma maneira conveniente de agrupar dados que pertencem a um único arquivo ou pacote de arquivos.
UIWindow	Objeto que coordena a apresentação de uma ou mais views na tela. A maioria dos aplicativos possuem apenas uma janela, a qual apresenta o conteúdo na tela principal, mas aplicativos podem ter janelas adicionais para mostrar o conteúdo em dispositivos externos.
View, controle e layer	Views e controles provêm a representação visual do conteúdo do aplicativo. Uma view é um objeto que desenha o conteúdo em uma designada área retangular e responde a eventos dentro dessa área. Um controle é um tipo especializado de view responsável por implementar objetos de interface como butões, áreas de texto, etc.

Tabela 5: Propriedades obtidas após processamento

parte das interações com os objetos da UIKit não definem o aplicativo mas ajudam a refinar seu comportamento.

A.3 Laço principal de execução

Ainda segundo o (APPLE, 2014a), o laço principal de execução de um aplicativo processa todos os eventos relacionados ao usuário. O objeto de UIApplication configura o laço principal de execução em tempo de abertura e o usa para processar eventos e gerenciar atualizações em interfaces relacionadas à view. Como o nome sugere, o laço principal de execução é executado na thread principal do aplicativo. Esse comportamento garante que eventos relacionados ao usuário serão processados serialmente na ordem em que forem recebidos.

A Figura 36 mostra a arquitetura do laço principal de execução e como os eventos do usuário resultam em ações tomadas pelo aplicativo. Com a interação do usuário com o

dispositivo, eventos relacionados a essas interações são gerados pelo sistema e entregues ao aplicativo através de uma porta especial configurada pelo UIKit. Eventos são sequenciados em fila internamente pelo aplicativo e disparados uma a uma no laço principal para serem executados. O objeto de UIApplication é o primeiro objeto a receber o evento e fazer a decisão sobre o que precisa ser feito. Um evento de toque é normalmente disparado para o objeto de janela principal, o qual por sua vez o dispara para a view em que o toque ocorreu. Outros eventos podem tomar caminhos levemente distintos através de outros objetos do aplicativo.

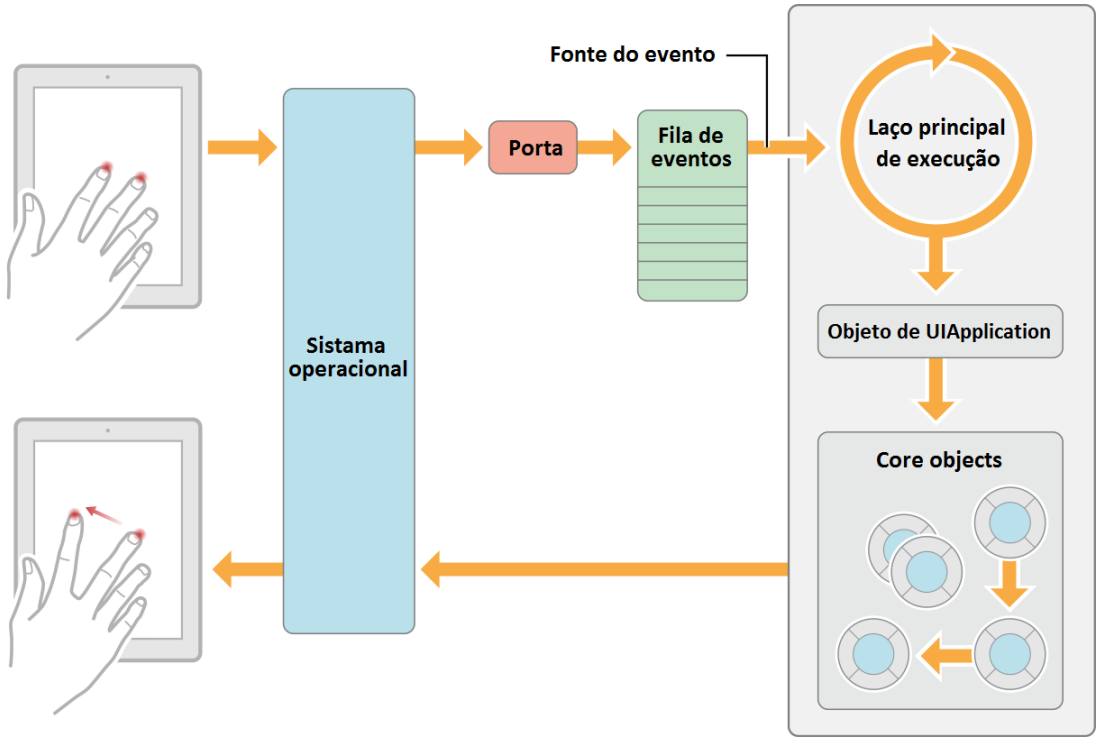


Figura 36: Processando eventos no laço principal de execução
Fonte: Adaptado de (APPLE, 2014a)

Muitos tipos de eventos podem ser entregues em um aplicativo iOS. Os mais comuns deles estão listados na Tab. (6). Muitos destes eventos são entregues usando o laço principal de execução do aplicativo, mas outros não. Alguns eventos são enviados ao objeto de delegate ou passados para um bloco provido pelo programador.

Alguns eventos, tais como os eventos de toque e controle remoto, são tratados pelos objetos do tipo responder do aplicativo. Objetos responder estão em todos os lugares do aplicativo - UIApplication, objetos de view e objetos de controle são todos exemplos de objetos do tipo responder. A maioria dos eventos tem como alvo um responder em específico mas podem ser passados para outros objetos responder (através da cadeia de resposta) se precisarem tratar um evento. Por exemplo, uma view que não trata um evento pode passá-lo para sua superview ou para a view controller.

Eventos de toque ocorrendo em controles (como um botões) são tratados diferen-

Tipo de evento	**Entregue a**	**Notas**
Toque	A view em que o evento ocorreu	Views são objetos do tipo responder. Um responder é um tipo de objeto que responde a eventos e os trata. Quaisquer eventos de toque não tratados pela view são repassados para o próximo elemento na cadeia de resposta.
Controle remoto e movimento de agito	Primeiro responder	Eventos de controle remoto são para controle de reprodução de mídia e são gerados por headphones e outros acessórios.
Acelerômetro, magnetômetro e giroscópio	Ao objeto designado	Eventos relacionados aos hardware de acelerômetro, magnetrômetro e giroscópio são entregues ao objeto que o desenvolvedor designar.
Localização	Ao objeto designado	O desenvolvedor registra para que receba eventos de localização usando o framework de Core Location.
Redesenhar	View a ser atualizada	Eventos de redesenho não envolvem objetos do tipo evento e são apenas chamadas para que a view se redesenhe.

Tabela 6: Tipos de evento comum em aplicativos iOS

temente de eventos de toque ocorrendo em outros tipos de view. Existem tipicamente um numero limitado de operações possíveis com uma controladora, e por isso essas interações são reagrupadas em mensagens de ação e entregues ao objeto destino apropriado. Esse padrão de projeto alvo-ação facilita o uso de controladoras para disparar a execução de código customizado no aplicativo.

A.4 Estados da aplicação

Como demonstrado por (APPLE, 2014a), em qualquer momento, um aplicativo sem encontra em um dos estados descritos na Tab. (7).O sistema move os aplicativo entre os estados em resposta a ações acontecendo em todo o sistema. Por exemplo, quando o usuário pressiona o botão de Home, um chamada telefônica é recebida, ou quaisquer outras interrupções ocorrem, o estado do aplicativo em execução é mudado em resposta. A Figura (37) mostra os caminhos que um aplicativo toma ao transitar entre os estados.

A maior parte das transições de estados são acompanhadas de uma chamada correspondente no objeto de delegate do aplicativo. Esses métodos são a maneira que o desenvolvedor tem de responder a mudanças no estado do aplicativo de maneira apropriada. Os métodos, assim como sua utilização esperada, estão listados abaixo.

- application:willFinishLaunchingWithOptions: - Este método é a primeira oportuni-

dade de executar código em tempo de lançamento.

- application:willFinishLaunchingWithOptions: - Este método permite realizar quaisquer código de inicialização antes de ser mostrado ao usuário.
- applicationDidBecomeActive: - Permite que o aplicativo saiba que está prestes a entrar em foreground.
- applicationWillResignActive: - Permite que o aplicativo saiba que que está movendo para fora do foreground. Método utilizado para colocar a aplicação em um estado quiescente.
- applicationDidEnterBackground: - Permite que o aplicativo saiba que está em background e pode ser suspenso a qualquer momento.
- applicationWillEnterForeground: - Permite que o aplicativo saiba que está saindo de background e entrando em foreground, mas não está ativo ainda.
- applicationWillTerminate: - Permite que o aplicativo saiba que será terminado. Este método não é chamado se o aplicativo estiver suspenso.

Estado	**Descrição**
Não executando	O aplicativo não foi lançado ou estava executando e foi terminado pelo sistema.
Inativo	O aplicativo está rodando em foreground mas não está recebendo eventos (mas pode estar executando código). Um aplicativo geralmente permanece nesse estado brevemente enquanto transita para outro estado.
Ativo	O aplicativo está rodando em foreground e está recebendo eventos. Este é o estado normal de um aplicativo em utilização em foreground.
Background	O aplicativo está em background e executando código. A maioria dos aplicativos permanece neste estado brevemente enquanto não são suspensos. Contudo, um aplicativo que requisita mais tempo de execução pode permanecer neste estado por um período de tempo. Adicionalmente, um aplicativo sendo lançado diretamente em background entra neste estado ao invés do estado inativo.
Suspenso	O aplicativo está em background mas não está executando código. O sistema move os aplicativos para este estado e não os notifica antes de fazê-lo. Enquanto suspenso, o aplicativo permanece em memória mas não executa nenhum código. Quando ocorre uma condição de baixa memória disponível, o sistema pode terminar aplicativos em suspensão sem aviso para liberar mais espaço para o aplicativo em foreground.

Tabela 7: Estados do aplicativo

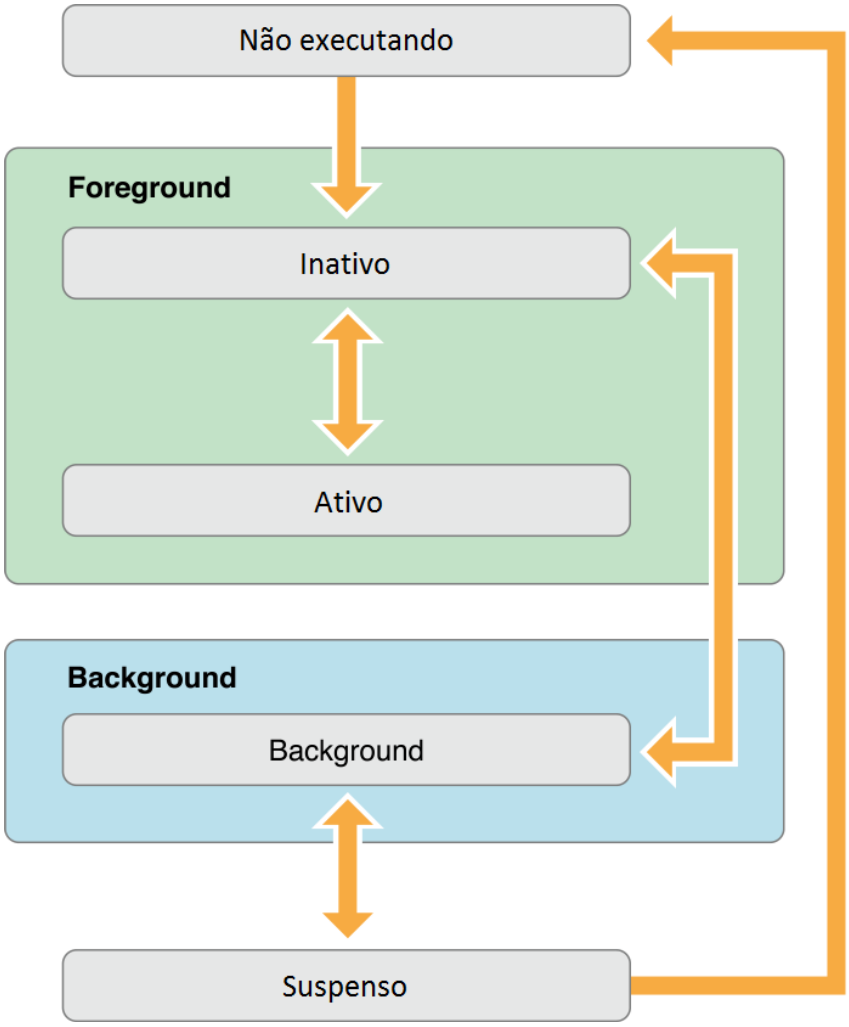


Figura 37: Mudanças de estado em um aplicativo iOS
Fonte: Adaptado de (APPLE, 2014a)

A.4.1 Finalização do aplicativo

Aplicativos devem estar preparados para serem terminados a qualquer momento e não devem esperar para salvar os dados do usuário ou executar quaisquer outras operações críticas. A finalização iniciada pelo sistema é parte normal do ciclo de vida do aplicativo. O sistema geralmente termina aplicativos para que possa recuperar memória e gerar espaço para outros aplicativos sendo lançados pelo usuário, mas o sistema também pode terminar aplicativos que não estão se comportando de maneira esperada ou não estão respondendo a eventos de maneira oportuna.

Aplicativos suspensos não recebem notificação quando são terminados; o sistema mata o processo e recupera a memória correspondente. Se um aplicativo está rodando em background e não está suspenso, o sistema chama o applicationWillTerminate: do delegate do aplicativo antes de ser terminado de fato. O sistema não chama esse método quando o dispositivo reinicia.

Em adição ao sistema terminar a aplicação, o usuário pode explicitamente terminar

um aplicativo utilizando a interface de multitarefa. O término iniciado pelo usuário tem o mesmo efeito que terminar um aplicativo suspenso. O processo do aplicativo é morto e nenhuma notificação é enviada ao aplicativo.

A.5 Threads e concorrência

O sistema cria o aplicativo na thread principal e o desenvolvedor pode criar threads adicionais, caso necessário, para realizar outras tarefas. Para aplicativos iOS, a técnica preferida é usar Grand Central Dispatch (GCD), objetos de operação, e outras interfaces de programação assíncrona ao invés de criar e manejar threads manualmente. Tecnologias como o GCD permitem que o desenvolvedor defina o trabalho que quer realizar e em que ordem quer realizá-lo, mas deixa que o sistema decida qual a melhor maneira para executá-lo nas CPUs disponíveis. Deixar que o sistema trate o gerenciamento de thread simplifica o código a ser escrito, torna mais fácil assegurar que o código está correto, e oferece melhor performance de maneira geral (APPLE, 2014a).

Ao trabalhar com threads e concorrência, os seguintes pontos devem ser considerados:

- Tarefas envolvendo views, Core Animation, e várias outras classes do UIKit normalmente devem ocorrer na thread principal do aplicativo. Existem exceções a esta regra, como manipulações de imagem que podem ocorrer em background para não afetar a fluidez da interface gráfica, mas de maneira geral se deve assumir que as tarefas envolvendo os itens acima devem ocorrer na thread principal do aplicativo.

- Tarefas que consomem muito tempo (o que são potencialmente grandes) devem sempre ocorrer em background. Quaisquer tarefas envolvendo acesso a rede, acesso a arquivos, ou grande quantidades de processamento de dados devem todas ser realizadas assincronamente utilizando o GCD ou objetos de operação.

- Em tempo de lançamento, as tarefas devem ser movidas para fora da thread principal sempre que possível. Em tempo de lançamento, o aplicativo deve usar o tempo disponível para preparar a interface do usuário o mais rápido possível. Somente tarefas que contribuem para o preparo da interface devem ser executados na thread principal. Todas as outras tarefas devem ser executadas assincronamente, com seu resultado sendo mostrado ao usuário assim que estiver prontos.